

44

LE JOURNAL QUI N'A PAS PEUR D'ETRE PIRATÉ

AVRIL 1988

BUG-
-BUSTERS

EDITORIAL

Oui, JEDI n'a pas peur des pirates. Les petits malins qui font l'acquisition de TURBO-Forth à un prix frisant l'indécence, pour le revendre, ne portent tort qu'à eux-même et à ceux auquels ils le cèdent. En effet, nous ne gagnons pas notre vie avec ce langage, alors quel préjudice subirions-nous? Non, celui qui prend TURBO-Forth légalement, qui est adhérent de JEDI et qui pose régulièrement des questions dans le FORUM est assuré d'un suivi qualitatif qui ne peut être assumé auprès du possesseur d'une version piratée. Celui-ci ne connaîtra pas l'existence de JEDI et ne sera jamais informé des dernières nouveautés de TURBO-Forth.

Et des nouveautés il y en a. Pour commencer, une virgule flottante mixte TURBO-PASCAL/TURBO-FORTH. Ensuite, le multi-fenêtrage et pour couronner le tout, un simulateur de processeur RCA 1802: exécution de code machine étranger au 8088 sur un PC ou compatible (programme adaptable à d'autres processeurs).

Ensuite, pour vous informer plus abondamment, JEDI fait maintenant 26 pages: 18 à 20 pages en français; 6 à 8 pages en "version originale" provenant de revues étrangères consacrées au FORTH. Bien entendu, si les articles en français deviennent abondants (...par ici les disquettes...), nous privilégierons leur diffusion.

SOMMAIRE

VERSION ORIGINALE

Nos coordonnées: ASSOCIATION JEDI, 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire: (1) 46.56.33.67 (attention, changement prochain de n°)

Téléchargement et Forum FORTH: 3615 SAM*JEDI

Le serveur SAM est la propriété de la Sté VICTEL, PARIS, commission paritaire SEQUOIA PRESSE.

TURBO-Forth et TURBO-Pascal

VIRGULE FLOTTANTE

Pour : PC et COMPATIBLES
Diffusion : 3615 SAM*JEDI FLOAT.FTH (le programme source en PASCAL est intégré à FLOAT.FTH après EOF; la version compilée du programme PASCAL est disponible dans le fichier FLOAT.COM). Ce programme sera également disponible dans le module M4 TURBO-Forth, en cours de remplissage.

Je vous adresse ce mois-ci un programme permettant l'utilisation des nombres en virgule flottante en TURBO-FORTH (quelques adaptations seront nécessaires pour F83 Laxen & Perry, si des gens l'utilisent encore...).

La méthode retenue est l'utilisation d'un programme résident en Turbo-PASCAL auquel on sous-traite les calculs ou les conversions à effectuer. En effet:

- pourquoi réinventer la poudre?
- on est sûr d'avoir des routines "bug-free";
- adaptation facile pour la version 4.0 de Turbo-PASCAL qui gère le coprocesseur s'il est là.

J'ai cherché une relative optimisation de la vitesse, c'est pourquoi de nombreuses portions de code apparaissent; mais il est vrai que FORTH permet ceci très facilement.
Malheureusement, l'absence de traitement des erreurs d'éxecution du Turbo-PASCAL alourdit beaucoup le programme PASCAL (traitement "manuel" des divisions par zéro, des dépassements trigo, etc...); l'erreur d'overflow ou d'underflow est irrécupérable et renvoie au DOS.

La création d'un programme résident en Turbo-PASCAL est fondée sur l'utilisation de l'interruption DOS 27H. Les lecteurs intéressés pourront se rapporter aux deux bons articles sur ce sujets parus dans "La Revue de l'Utilisateur de l'IBM PC" des mois de décembre 1987 et janvier 1988.

L'interruption choisie pour l'activation du programme PASCAL résident est la 44H, dont le vecteur est en 0000:110

- 0000:113 (hex).

On vérifiera sur sa configuration que cette interruption est libre (il doit y avoir des zéros dans cette zône mémoire) ainsi que la suivante. En effet, le programme d'installation (procédure INSTALL) va ranger en 0000:114-117 l'offset et le segment du début de la zône des variables, données indispensables au programme FORTH pour l'échange de paramètres.

Le fonctionnement est très simple: pour faire effectuer une tâche à PASCAL, FORTH va initialiser les variables PASCAL nécessaires, par exemple x et y pour une addition, va inscrire un entier dans la variable PASCAL Operation et appeler l'interruption 40H. Au retour, FORTH lit les résultats (dans x, n ou sx selon le cas) ainsi que le contenu de la variable erreur pour ABORT éventuel. A noter que l'affichage des flottants n'est pas sous-traité à PASCAL ('writeln'); en effet, il est indispensable que tous les affichages en FORTH passent par EMIT, en cas de redirection éventuelle.

Le listing du programme PASCAL n'appelle pas de commentaire particulier.

Pour le programme FORTH, je n'ai repris du F-PACK.FTH de M. ZUPAN que quelques mots de manipulation de pile. J'ai redéfini F@ et F! en code. J'ai introduit F, et (FLIT) pour des raisons d'homogénéité avec la compilation des entiers simple ou double précision, mais il faudra que le Cher Secrétaire rajoute un cas particulier à son décompilateur (mot SEE) (Ndlr: que de travail on exige de moi...!).

Un mot sur l'utilisation: un flottant s'introduit précédé du mot & : j'ai cherché un mot d'une lettre encore non utilisé et celui-ci me paraît tout indiqué (pour ceux qui feront le résident en Turbo-C gérant les flottants double précision, ils utiliseront tout naturellement &&...).

Un flottant utilise 6 octets. Ces octets sont stockés en mémoire en FORTH dans les constantes ou les variables dans le même ordre qu'en PASCAL; sur la pile, le dernier mot (deux octets) empilé est celui correspondant à l'adresse basse (voir la définition code de F! pour s'en convaincre).

Le mot N.M (S n m --) gère le formattage de l'affichage des flottants: champ de n caractères justifiés à droite, avec m chiffres derrière le point décimal. Si m est strictement supérieur à 24, le format en virgule flottante sera utilisé. Le reste des mots est d'usage et d'utilisation évidents. Attention: INT renvoie un résultat flottant; faire suivre de ROUND si nécessaire.

Un dernier mot sur les performances: la ligne

```
 :: TIME
 &2 100 0
 DO &2 F* LOOP   100 0
 DO &2 F/ LOOP   CR F. TIME   CR .INTERVAL ;
```

affiche 2.0000000 et 00:00:00,22 sur un AMSTRAD PC1512.

LISTING PASCAL: (intégré à FLOAT.FTH)

```
PROGRAM RESIDENT_YGS_PAS;

CONST
     save_DS: integer= $0000;

type
     enreg = record case integer of
    1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS :integer);
    2:(AL,AH,BL,BH,CL,CH,DL,DH:BYTE);
            end;
var
     x,y : real;
     sx  : string[18];
     erreur, operation, n : integer;
     resultat : boolean;
     ndigits, mdigits : integer;
     offs: integer absolute $0000:$0114;
     { offset debut zône des variables }
     segm: integer absolute $0000:$0116;
     regs: enreg;
     vect40_offset:integer absolute $0000:$0110;
     vect40_segment:integer absolute $0000:$0112;
     last_var : byte;

const
     pi : real = 3.1415926538979;


PROCEDURE PT_ENTR;
{---------------------}
Begin
     with regs do
     begin   {Empilage des registres internes.}
           inline($50/$53/$51/$52/$56/$57/$1E/$06);
           inline($FA);
           inline($2E/$8E/$1E/save_DS);
           inline($FB);

{-------------TRAITEMENT---------------}

       erreur := 0;
       case operation of
          1: val(sx,x,erreur);
          2: x := n;
          3: x := y+x;
          4: x := y-x;
          5: x := y*x;
          6: if x = 0 then erreur := 1 else x := y/x;
    7: if y <= 0 then erreur := 2 else x := exp(x*ln(y));
          8: resultat := y=x;
          9: resultat := y<>x;
         10: resultat := y>x;
         11: resultat := y<x;
         12: resultat := y>=x;
```

```
          13: resultat := y<=x;
          14: if x = 0 then erreur := 1 else x := 1/x;
          15: x := -x;
          16: x := abs(x);
          17: x := sin(x);
          18: x := cos(x);
 19: if cos(x) = 0 then erreur := 3 else x := sin(x)/cos(x);
 20: if (x<-1) or (x>1) then erreur := 4 else
       begin
          if x = 1 then        { arcsin }
                x := pi/2
          else
                x:=arctan(x/sqrt(1-sqr(x)));
          end;
 21: if (x<-1) or (x>1) then erreur := 4 else
       begin
          if x <> 0 then       { arccos }
             begin
                if x = -1 then
                   x := -pi
                else
                   x:=arctan((sqrt(1-sqr(x))/x));
                   if x<0 then x := x+pi;
                      end
                   else x := pi/2;
                   end;
 22: if (x<-1) or (x>1) then
              erreur := 4
          else x := arctan(x);
          23: x := int(x);
          24: x := frac(x);
          25: x := sqr(x);
          26: if x<0 then
   erreur := 5 else x := sqrt(x);
            27: if x<=0 then
   erreur := 6 else x := ln(x);
            28: if x<=0 then
   erreur := 6 else x := ln(x)/ln(10);
            29: x := exp(x);
       30: str(x:ndigits:mdigits,sx);
       31: if (x<-32767) or (x>32767) then
            erreur := 7 else n := round(x);
      end;


{---------FIN DU TRAITEMENT---------------}

            inline($FA);
            inline($FB);
            inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
            inline($CF);
       end;
end;

   PROCEDURE INSTALL;
{ --------------------  }

Begin
  ndigits := 0;
  mdigits := 4;
  with regs do
  begin
       inline($FA);
       vect40_segment:= cseg; { Adr. de base pt. d'entrée }
       vect40_offset:= ofs(pt_entr)+7;
                              { Offset pt. d'entrée       }
       inline($FB);
       save_DS:= dseg;          { sauve seg. données        }
       segm:= seg(x);           { transfert adresse du début }
       offs:= ofs(x);           { de la zône des variables  }
       writeln;
       writeln('Programme installé !! ');
       writeln;
       AX := $3100;
       DX := Dseg - Cseg + ((ofs(last_var) div 16)) + 1;
       intr($21,regs);
  end;
end;

BEGIN
  writeln;
  if vect40_segment <> ofs(pt_entr)+7 then
     { Test déjà installé?  }
       install
    else
    begin
       writeln;
       writeln('installation déjà effectuée.');
       writeln;
    end;
END.
```

LISTING FORTH: FLOAT.FTH

```
\ ***************************************************************
\                      VIRGULE FLOTTANTE
\ ***************************************************************
\
ECHO OFF
ONLY FORTH ALSO DEFINITIONS DECIMAL
VARIABLE PSEG
   \ Contiendra le segment du programme PASCAL résident
VARIABLE AREA
   \ Offset de la zône des variables PASCAL
18 STRING SX$

HEX
44 CONSTANT FINT     \ Interruption
: INIT (S -- )
  " Program FLOAT.COM" $EXECUTE
  0 116 L@ PSEG !
  0 114 L@ AREA ! ;
INIT

\ Variables et constantes
CODE F! (S real add -- )
     DI POP   AX POP   AX STOS
     AX POP   AX STOS   AX POP   AX STOS
     NEXT  END-CODE

CODE F@ (S ad -- real )
     SI W MOV      \ Sauvegarde SI
     SI POP   AX LODS   AX DX MOV   AX LODS
     AX CX MOV   AX LODS   AX PUSH   CX PUSH
     DX PUSH   W SI MOV   \ On rétablit SI = IP
     NEXT  END-CODE

: F, ( n1 n2 n3 ---)
    , , , ;
: FVARIABLE
    CREATE 0  0  0  F,  DOES> ;
: FCONSTANT
    CREATE HERE 6 ALLOT F! DOES> F@ ;

FVARIABLE FST1     FVARIABLE FST2     FVARIABLE FST3

\ Manipulations de pile
: FDUP
    3DUP ;
: FOVER
    5 PICK 5 PICK 5 PICK ;
: FROT
     FST1 F! FST2 F! FST3 F!
     FST2 F@ FST1 F@ FST3 F@ ;
: FSWAP
     FST1 F! FST2 F! FST1 F@ FST2 F@ ;
: FDROP
     2DROP DROP ;

\ Routines élémentaires de transfert de données.
ASSEMBLER
LABEL PUTX
      PSEG #) ES MOV    AREA #) DI MOV
      21 # DI ADD
          \ ES:DI pointe vers la variable OPERATION
      DX AX MOV          \ Code opération dans AX
      AX STOS            \ Code opération stocké
      AREA #) DI MOV     \ DI pointe vers la variable X
   HERE                  \ Label PUTY, compilé plus tard
      W POP
          \ Sauvegarde adresse de retour du RET
      AX POP   AX STOS   AX POP
      AX STOS   AX POP   AX STOS   W PUSH   RET
CONSTANT PUTY

LABEL GETX
```

```
      SI DX MOV            \ Sauvegarde SI
      AREA #) SI MOV   PSEG #) DS MOV
      1F [SI] AX MOV      \ Récupération code d'erreur
      0 # AX CMP
      0=
      IF                   \ Si pas d'erreur...
         STD
         4 # SI ADD
         \ DS:SI pointe vers le dernier mot de X
         AX LODS
         AX PUSH    AX LODS
         AX PUSH         \ Empilage de X
         AX LODS    AX PUSH
         CLD
         \ Pour le bon fonctionnement de l'interprêteur
         FALSE # AX MOV \ Drapeau
      THEN
      DX SI MOV            \ On rétablit SI
      CS DX MOV            \ On rétablit DS
      DX DS MOV    1PUSH

\ Cconversion chaîne de caractères --> flottant
: VAL (S ad len -- real_if_ok )
  12 MIN                  \ 18 caractères maximum
  ASM[
      PSEG #) ES MOV   AREA #) DI MOV
      0C # DI ADD
      \ ES:DI pointe vers la variable SX du PASCAL
      CX POP              \ Longueur...
      CX AX MOV
      AL BYTE STOS        \ ...dans le premier octet de SX
      SI DX MOV           \ Sauvegarde SI
      SI POP
      \ DS:SI pointe vers le premier caractère
      \ de la chaîne
      REP BYTE MOVS       \ Transfert FORTH --> PASCAL
      1 # AX MOV    AREA #) DI MOV
      21 # DI ADD
      \ ES:DI pointe vers la variable OPERATION
      AX STOS             \ Opération := 1
      DX SI MOV           \ On rétablit SI
      FINT INT    GETX #) JMP
  ]FORTH ABORT" Erreur de conversion !" ;

\ Conversion flottant - chaîne de caractères - Affichage
CODE FSTR$ (S real -- add len )
      1E # DX MOV         \ Code opération
      PUTX #) CALL
      FINT INT    SX$ DROP 1 - # DI MOV
      CS AX MOV
      AX ES MOV           \ ES:DI pointe vers SX
      SI W MOV            \ Sauvegarde SI
      AREA #) SI MOV   0C # SI ADD
      PSEG #) DS MOV
      AX BYTE LODS        \ Longueur dans AL...
      AX BYTE STOS    AH AH XOR    AX CX MOV
      AX DX MOV           \ ...et dans DX
      REP BYTE MOVS       \ Transfert
      W SI MOV            \ On restaure SI
      CS AX MOV    AX DS MOV
      SX$ DROP # AX MOV    AX PUSH
      DX PUSH    NEXT
END-CODE

: F. (S real -- )
  FSTR$ TYPE ;

: F? (S real -- real )   \ Affichage non destructif
  FDUP F. ;

: N.M (S n m -- )
  PSEG @ AREA @ 28 + L!
  PSEG @ AREA @ 26 + L! ;

\ Mots de définition
: ?FERROR (S err_code -- )
  ?DUP
  IF
     CASE
      1 OF " Division par zéro!" ENDOF
      2 OF " x^y avec x<0!" ENDOF
      3 OF " Tangente infinie!" ENDOF
      4 OF " Hors limites pour Arcsin ou Arccos!" ENDOF
      5 OF " Racine carrée d'un nombre négatif!" ENDOF
      6 OF " Logarithme d'un nombre négatif ou nul!" ENDOF
     ENDCASE TRUE ?ERROR
  THEN ;

: MONADIC (S op -- )
  CREATE C,
  DOES> (S real -- real )
  C@
  ASM[
      DX POP              \ Code opération dans DX
      PUTX #) CALL   FINT INT   GETX #) JMP
  ]FORTH ?FERROR ;

: DYADIC (S op -- )
  CREATE C,
  DOES> (S real real -- real )
  C@
  ASM[
      DX POP              \ Code opération dans DX
      PUTX #) CALL    PUTY #) CALL
      FINT INT   GETX #) JMP
  ]FORTH ?FERROR ;

: COMPARISON (S op -- )
  CREATE ,
  ;CODE   (S real real -- t/f )
      W INC    W INC
      0 [W] DX MOV        \ Code opération dans DX
      PUTX #) CALL    PUTY #) CALL
      FINT INT    PSEG #) ES MOV
      AREA #) W MOV    ES: 25 [W] AL MOV
      \ On récupère le contenu de la variable RESULTAT
      AH AH XOR   1PUSH
  END-CODE

\ Opérations en virgule flottante
DECIMAL
 3 DYADIC F+        4 DYADIC F-
 5 DYADIC F*        6 DYADIC F/
 7 DYADIC F^

 8 COMPARISON F=     9 COMPARISON F<>
10 COMPARISON F>    11 COMPARISON F<
12 COMPARISON F>=   13 COMPARISON F<=

14 MONADIC INV      15 MONADIC FNEGATE
16 MONADIC FABS     17 MONADIC SIN
18 MONADIC COS      19 MONADIC TAN
20 MONADIC ASIN     21 MONADIC ACOS
22 MONADIC ATAN     23 MONADIC INT
24 MONADIC FRAC     25 MONADIC ^2
26 MONADIC SQR      27 MONADIC LN
28 MONADIC LOG      29 MONADIC EXP

HEX
: ROUND (S real -- n )
  ASM[
     1F # DX MOV    PUTX #) CALL
     FINT INT    AREA #) W MOV
     ES: 23 [W] AX MOV    \ Résultat
     AX PUSH
     ES: 1F [W] AX MOV    \ Code d'erreur
     1PUSH
  ]FORTH " Dépassement entier!" ROT ?ERROR ;

: FLOAT (S n -- real )
ASM[ W DX MOV
     AREA #) W MOV    PSEG #) ES MOV
     AX POP               \ On récupère n...
     ES: AX 23 [W] MOV    \ ... que l'on transmet au PASCAL
     2 # AX MOV           \ Code opération...
     ES: AX 21 [W] MOV    \ ... stocké
     DX W MOV    FINT INT    GETX #) JMP
  ]FORTH  DROP ;  \ Pas d'erreur possible dans ce cas
DECIMAL

\ Mot &
CODE (FLIT)
     AX LODS    AX DX MOV
     AX LODS    AX CX MOV
     AX LODS    AX PUSH
     CX PUSH    DX PUSH    NEXT
```

```
END-CODE

: FLITERAL
  STATE @
  IF
     COMPILE (FLIT) F,
  THEN ; IMMEDIATE

: &            \ (S < real > -- real )
  BL WORD COUNT VAL [COMPILE] FLITERAL ;  IMMEDIATE

\ Constantes utiles...
& 3.1415926538979 FCONSTANT PI
& 0 FCONSTANT &0
& 1 FCONSTANT &1
& 2 FCONSTANT &2
& 1 EXP FCONSTANT &E
```

COURRIER:

- A plusieurs occasions sont apparus dans JEDI des bouts de module traitant des structures CASE... améliorées. J'avais moi-même déjà pondu quelque chose dans ce genre où je traite en plus des ensembles. A ce propos, je ne suis pas d'accord pour employer le mot ensemble à la place d'intervalle, cf JEDI 42. Je vous envoie donc mes résultats.
- A propos du serveur SAM*JEDI: pas de problème ça marche, mais les fichiers ASCII ne pourraient-ils pas être compactés par un utilitaire comme ARC que tous ceux qui font du téléchargement sur PC disposent (?). De plus, serait-il possible de savoir quelle version il faut utiliser de TURBO pour pouvoir les compiler. Exemple: pour le programme de compactage de M. ZUPAN; il utilise deux mots START et >VIEW qui ne sont pas dans ma version de TURBO mais sont dans le F-83 de Laxen et Perry. Ca m'étonnerait que Michel utilise cette versio car il est aux premières loges pour avoir la dernière version.
- Quelqu'un aurait-il un désassembleur 8086-88 en Forth! Je suis en train de m'en faire un, mais comme en même temps j'apprends l'assembleur: ça ne va pas vite.

J.L. SIRET 72460 SAVIGNE L'EVEQUE

REPONSE:
- Merci pour votre version améliorée de CASE. Elle est diffusée un peu plus loin.
- Nous diffusons les programmes en clair sur SAM*JEDI, du moins en ce qui concerne les fichiers source: pas tout le monde dispose du programme ARC (essayez de le trouver en disk 3'1/2 pour OLIVETTI PC ou AMSTRAD portable...). Un fichier ASCII peut être capturé par un autre système qu'un PC ou compatible; en effet, TURBO-Forth peut être implanté sur d'autres systèmes. Enfin, SAM*JEDI est le seul serveur à diffuser des programmes en "freeware" et en code source. Tout les autres diffusent en compacté et seulement en exécutable et parfois nécessitent un utilitaire particulier d'éclatement et de décompactage (ex: 3616 VIFI). Nota: le temps de transfert d'un fichier indiqué par SAM*JEDI est surestimé; il faut pratiquement le diviser par quatre; petite erreur qui sera corrigée en temps voulu.
- Les fichiers sont exécutable avec la seule version de TURBO-Forth existante à ce jour: la version définitive. TURBO est maintenant bétonné: toute modification, amélioration ou correction sera signalée dans JEDI (exemple JEDI 42, page 18).

LISTING:

```
\ ***************************************************************
\   Mise en oeuvre de structures de contrôle améliorées
\                à la façon PASCAL
\ ***************************************************************
ONLY FORTH DEFINITIONS DECIMAL
\ Pour mettre en oeuvre un intervalle:
\   2PICK sert à récupérer le nombre à tester
: 2PICK >R >R DUP R> R> ;
: ..OF COMPILE 2PICK COMPILE BETWEEN COMPILE ?BRANCH
   >MARK COMPILE DROP ; IMMEDIATE

\ Pour mettre en oeuvre limité par un seul nombre
\ et l'infini négatif, les autres possibilités se feraient
\ de même en remplaçant < par > ou >= etc...
: <OF   COMPILE  OVER   COMPILE  -ROT  COMPILE  < COMPILE
?BRANCH
   >MARK COMPILE DROP ; IMMEDIATE

\ Pour mettre en oeuvre les ensembles
\  (valeurs discètes continues)
\ 1ère façon les nombres restent s ~ la pile: lourd à
\ écrire s'il y en a beaucoup
VARIABLE PILE : {{ SP@ PILE ! ;
: }} PILE @ SP@ - 2/ DUP PICK SWAP 1- 0 0 -ROT
   ?DO >R DUP ROT = R> OR LOOP NIP ;
: }}OF COMPILE }} COMPILE ?BRANCH >MARK
   COMPILE DROP ; IMMEDIATE

\ 2ème façon où les valeurs sont contenues dans un tableau
\ 8 bits. Ces valeurs sont entrées lors de la déclaration
: ENTREN BL WORD NUMBER DROP ; ( pour des nombres )
: ENTREA [COMPILE] ASCII ;  ( pour des codes ASCII)
: :CRETAB
   CREATE DUP C, 0 ?DO ENTREA C, LOOP
   DOES> COUNT ;
4 :CRETAB TABESS * / + -   ( par exemple)
: }}2  0 0 -ROT
   ?DO  I SWAP >R >R 2DUP R> + C@ = R> OR LOOP
   NIP ;
: }}OF2  COMPILE }}2 COMPILE ?BRANCH
   >MARK COMPILE DROP ; IMMEDIATE
EOF  \ Enlever EOF pour exécuter la suite
: ESSAI BEGIN KEY DUP . CASE
   ASCII A OF ." C'est un A " ENDOF
   ASCII B ASCII Z ..OF ." majuscule entre B et Z " ENDOF
   ASCII 0 ASCII 9 ..OF ." c'est un chiffre " ENDOF
   {{ ASCII + ASCII - ASCII * ASCII / }}OF
       ." c'est un signe d'opération "  ENDOF
   ASCII ° OF ." c'est un ° " ENDOF
   ASCII a ASCII z ..OF ." c'est une minuscule " ENDOF
   32 <OF ." code inférieur à 32 " ENDOF
   ." sûr que ça existe ça ? "
   ENDCASE  CR ASCII A = UNTIL ;

: ESSAI2 BEGIN KEY DUP DUP . CASE
   ASCII A OF ." C'est un A " ENDOF
   ASCII B ASCII Z ..OF ." majuscule entre B et Z " ENDOF
   ASCII 0 ASCII 9 ..OF ." c'est un chiffre " ENDOF
     TABESS }}OF2 ." c'est un signe d'opération "  ENDOF
   ASCII ° OF ." c'est un ° " ENDOF
   ASCII a ASCII z ..OF ." c'est une minuscule " ENDOF
   32 <OF ." code inférieur à 32 " ENDOF
   ." rien de répertorié "
   ENDCASE  CR ASCII A = UNTIL ;
\ NOTE: sur F83 et ATMOS il faut faire par exemple:
\ : ..OF 4 ?PAIRS puis de COMPILE 2PICK ... à DROP
\ puis mettre à la fin 5 ;
```

Turbo-PROLOG

CARNET D'ADRESSE A ACCES CODE

par Nicolas BRUN

pour : PC et compatibles, avec TURBO-Prolog
diffusion : 3615 SAM*JEDI, programme ADRS.PRO

ADRS.PRO est un carnet d'adresse doté d'une clé d'accès, permettant la sauvegarde sur disque des noms, prénoms, adresses et numéros de téléphone de vos relations ainsi que des commentaires sur ceux ci.

Lancer ADRS. L'écran devient rose et, le programme vous demande le code d'entré (123 pour la version compilée). Si vous utilisez le fichier source en turbo prolog, il vous seras facile de le remplacer par un autre (Voir le fichier source); si le code est faux, le programme s'interrompt. Une fois le code entré, l'écran devient vert et affiche le menu principal.

Le choix (1), permet de charger la base de données mais, quand ce fichier (ADRS.DBA) n'est pas dans le répertoire courant le programme s'arrête. En effet, il n'est pas possible de charger une base qui n'existe pas; celle ci est créée par la combinaison des fonctions (2) et (5).

ATTENTION, le chargement plusieurs fois de suite de la base de données pendant une meme séance de travail, duplique les données en mémoire puis dans votre fichier, ce qui mene vite à une saturation du programme.

CREATION DE LA BASE DE DONNEES:

Sélectionner (2), cette fonction permet d'ajouter des adresses à la base de données et de la créer. Les données sont d'abord stockées en mémoire puis sauvegardées sur disque lorsque l'on quitte le programme par la fonction (5). Un menu indique la marche a suivre; créez quelques fiches puis sélectionnez (6) pour retourner au menu principal.

Consulter la base de données: sélectionner (3); cette fonction recherche dans la base les adresses stockées. Un menu demande le nom et le prénom de la personne recherchée et renvoie son adresse. Essayez avec les noms que vous avez stockés puis sélectionnez (6) pour retourner au menu principal.

Effacer un élément de la base de données: cette fonction (4) procède de la même façon que la fonction de recherche, mais l'adresse trouvée est effacée de la base de données. Si vous effacez par mégarde un élément de la base et, que celle-ci a été sauvegardée, il est possible en quittant le programme de recupérer cette en renommant le fichier ADRS.BAK en ADRS.DBA)

Quitter le programme: la fonction (5) a plusieurs possibilités; elle permet la sauvegarde de la base de données sur le disque ou de la créer si elle n'existait pas. Elle permet aussi la sauvegarde de l'état précédent de la base dans un fichier de type BAK et permet enfin de quitter le programme.

ATTENTION, quand vous quittez le programme et que vous n'avez pas chargé la base auparavant, (si elle existait bien sûr), vous allez effacer le contenu du fichier ADRS.DBA. Si tel est le cas, une fois sorti du programme, renommez le fichier ADRS.BAK en ADRS.DBA.

LISTING:

ADRS.PRO

```
/* ADRS.PRO, carnet d'adresse a accès codé */

database
adresse(symbol,symbol,symbol,symbol,symbol)

predicates
     test(integer) /* ce predicat verifie la validite du code d'entrée */
     code /* celui la lance le programme */
     go /* celui ci affiche le menu principal */
     choix(integer) /* ce dernier sert a appeler les sous fonctions */
clauses

     code :-
          clearwindow,window_attr(79),
          cursor(5,21),
          write("Carnet d'adresse a accès protégé V 1.3 "),nl,
          cursor(6,21),
          write("          nicolas Brun, mai 1988"),
          cursor(10,21),
          write("Entrez votre numero d'identification: "),
          readint(X),test(X).
     test(X) :-
          X = 123 , go.
     test(X) :-
          X <> 123 , exit.
/* vous pouvez remplacer le code 123 par le votre pourvu qu'il soit compris
   entre -32768 et 32767 */

     go :
          clearwindow,window_attr(47),nl,nl,
          write("                    CONSULTATION DU CARNET D'ADRESSE "),
          nl,nl,nl,
          write("  (1) Charger la base (2) Ajouter (3) Chercher  (4) Effacer (5) Quitter"),
          nl,nl,nl,
          write("                          Entrez votre choix: "),readint(X),choix(X).
     go :-
          go.

     choix(X) :-
          X > 6,go.
     choix(X) :-
          X=0,go.
     choix(2) :-
          clearwindow,window_attr(71),
          write("Vous voulez AJOUTER une adresse a votre carnet"),
          nl,
          write(" validez chaqu'une de vos entrées par <RETURN>,"),
          nl,
          write("Entrez le Nom: "),
          readln(N),nl,
          write("Entrez le Prenom: "),
          readln(P),nl,
          write("Entrez l'Adresse: "),
          readln(A),nl,
          write("Entrez le numero de Telephone: "),
          readln(T),nl,
          write("Entrez les Commentaires: "),
          readln(C),nl,
          asserta(adresse(N,P,A,T,C)),
```

```
        write("  (2) Ajouter une autre adresse  (6) Menu principal "),
        nl,nl,
        write("  Entrez votre choix: "),
        readint(X),choix(X).
    choix(3) :-
        clearwindow,
        window_attr(23),
        write("Vous voulez CHERCHER une adresse dans votre carnet"),
        nl,
        write(" validez votre entrée par <RETURN>,"),
        nl,
        write("Entrez le Nom de la personne recherchée: "),
        readln(N),nl,
        write("Entrez le Prenom de la personne recherchée: "),
        readln(P),nl,
        adresse(N,P,A,T,C),
        clearwindow,
        write("L'adresse de: ",P," ",N," est: ")
        ,nl,nl,
        write(P," ",N),nl,
        write(A),nl,
        write("Telephone: ",T),nl,nl,
        write("Commentaire: ",C),nl,nl,
        write("  (3) Chercher une autre adresse  (6) Menu principal "),
        nl,nl,
        write("  Entrez votre choix: "),
        readint(X),choix(X).
    choix(3) :-
        clearwindow,
        write("Il n'existe pas de personne portant ce nom dans votre carnet"),
        nl,
        write("  (3) Chercher une autre adresse  (6) Menu principal "),
        nl,nl,
        write("  Entrez votre choix: "),
        readint(X),choix(X).
    choix(3) :-
        choix(3).
    choix(4) :-
        clearwindow,window_attr(87),
        write("Vous voulez EFFACER une adresse dans votre carnet"),
        nl,
        write(" validez votre entrée par <RETURN>,"),
        nl,
        write("ATTENTION, CE CHOIX EST IRREMEDIABLE"),nl,nl,
        write("Entrez le Nom puis le prenom de la personne que vous vouler effacer: "),
        readln(N),nl,readln(P),
        adresse(N,P,A,T,C),
        clearwindow,
        write("L'adresse effacée est: ")
        ,nl,nl,
        write(P," ",N),nl,
        write(A),nl,
        write("Telephone: ",T),nl,nl,
        write("Commentaires: ",C),nl,nl,
        retract(adresse(N,P,A,T,C)),
        write("  (4) Effacer une autre adresse  (6) Menu principal "),
        nl,nl,
        write("  Entrez votre choix: "),
        readint(X),choix(X).
    choix(4) :-
        clearwindow,
        write("Il n'existe pas de personne portant ce nom dans votre annuaire"),
        nl,
        write("  (4) Effacer une autre adresse  (6) Menu principal "),
        nl,nl,
        write("  Entrez votre choix: "),
        readint(X),choix(X).
    choix(4) :-
        choix(4).

    choix(5) :-
        system("del adrs.bak"), /* ce predicat effectue la maintenance */
        system("ren adrs.dba adrs.bak"),/* du fichier de données de la base */
        save("adrs.dba"),exit./* ainsi que sa creation quand il n'existe pas */

    choix(6) :-
        go.
    choix(1) :-
        consult("adrs.dba"),go. /* ce predicat charge la base de donné */
goal
    code
```

FORTH

MULTI-FENETRAGE TURBO-Forth

par Michel ZUPAN

pour : PC et compatibles
diffusion : 3615 SAM*JEDI
et module M4 TURBO-Forth (en préparation)

LISTING: WINDOWS.FTH

```
HEX
B800    800 40 10 LC@ 30 AND 30 = * +
     0  2CONSTANT VIDEO
( la configuration détermine la mémoire vidéo en)
( B800:0000 ou B000:0000 )
DECIMAL
80 CONSTANT #COL \ écran: nombre de colonnes
25 CONSTANT #LIN \ écran: nombre de lignes
VARIABLE WDO#     \ fenêtre courante
VARIABLE WIDE     \ largeur fenêtre courante
VARIABLE HIGH     \ hauteur fenêtre courante
VARIABLE VISILIN \ nombre de lignes à calquer
VARIABLE VISICOL \ nombre de colonnes à calquer
VARIABLE VISI-X   \ position colonne du calque
VARIABLE VISI-Y   \ position ligne du calque

: CORNER ( -- x y )
\ position du coin haut-gauche de la fenêtre
  WDO# @ 2+ DUP @ SWAP 2+ @ ;

: INTERVAL ( x l max -- x l )
\ primitive de découpe d'un calque
  >R OVER 0 MAX -ROT + R> MIN OVER - ;

: CACHE ( -- )
\ découpe d'un calque visible de la fenêtre courante
  CORNER HIGH @ #LIN INTERVAL VISILIN ! VISI-Y !
         WIDE @ #COL INTERVAL VISICOL ! VISI-X ! ;

: TRANSFER ( c1 seg1 adr1 c2 seg2 adr2 -- )
\ transfert de zones
      VISILIN @ 0> VISICOL @ 0> AND IF
      VISILIN @ 0 DO
      4 PICK 4 PICK 7 PICK I * 2* +
      3 PICK 3 PICK 6 PICK I * 2* +
      VISICOL @ 2* LCMOVE LOOP
      THEN 2DROP 2DROP 2DROP ;

: SIGHT ( -- cols seg adr ) \ zone d'affichage
      WIDE @ DSEGMENT
      WDO# @ 6 +
      CORNER VISI-Y @ SWAP - WIDE @ *
             VISI-X @ ROT - + 2* + ;

: VISIBLE ( -- cols seg adr )
\ zone affichable d'une fenêtre
      #COL VIDEO
      VISI-Y @ #COL * VISI-X @ + 2* + ;

: WDOBUF  ( -- cols seg adr )
\ zone tampon plan postérieur de SIGHT
      VISICOL @ DSEGMENT
      WDO# @ WIDE @ HIGH @ * 2* 6 + + ;

\ mots utilisateur
: WINDOW ( largeur hauteur -- ) \ définition d'une fenêtre
2 ?ENOUGH
CREATE OVER C, DUP C,  \ hauteur, largeur
       0 , 0 ,          \ position colonne,ligne
       OVER #COL MIN OVER #LIN MIN *
       -ROT * + 2*
       HERE OVER ALLOT SWAP BLANK
DOES>  DUP C@ WIDE ! DUP 1+ C@ HIGH ! WDO# ! CACHE ;

: AT# ( x y -- ) \ positionne en x,y la fenêtre courante
  WDO# @ TUCK 4 + ! 2+ ! CACHE ;

: OPEN# ( x y -- ) \ ouvre la fenêtre courante
      VISIBLE  WDOBUF TRANSFER
        SIGHT VISIBLE TRANSFER ;

: CLOSE# ( -- )      \ ferme la fenêtre courante
       WDOBUF VISIBLE TRANSFER ;

: TAKE# ( -- )      \ initialise la fenêtre avec l'écran
       VISIBLE SIGHT TRANSFER ;

EOF \ supprimer EOF pour exécuter les exemples
\ Démonstration de WINDOWS  (monochrome)
echo off
DECIMAL
: TEMPO 10000 0 DO LOOP ; \ ralentit les mouvements
: ATTFILL# ( att -- ) \ remplit  la fenêtre  courante avec
attribut att
  WDO# @ 5 + WIDE @ HIGH @ * 0 DO 2+ 2DUP C! LOOP 2DROP ;

40 11 WINDOW WIN1
DARK
   .( ╔══════════════════════════════════════╗)
CR .( ║ Voici la fenêtre d'essai n° 1        ║)
CR .( ║                                      ║)
CR .( ║ Une fenêtre est un espace affichable ║)
CR .( ║ qui se superpose à l'écran quand  on ║)
CR .( ║ l'ouvre  et  qui  restitue  celui-ci ║)
CR .( ║ quand on la ferme.                   ║)
CR .( ║                                      ║)
CR .( ║ Une fenêtre  peut  être  positionnée ║)
CR .( ║ à n'importe quel endroit de l'écran. ║)
CR .( ╚══════════════════════════════════════╝)
WIN1 TAKE#
:: 15 0 DO I 2* I AT# OPEN# TEMPO CLOSE# LOOP ; OPEN#
128 5 WINDOW WIN2
0 0 AT
.( ┌───────────────────────────────────────)
CR .( │ Fenêtre  n°2 :  128 colonnes  / 5 lignes : partie
gauche       )
CR .( │ Une fenêtre peut avoir des  dimensions supérieures
à celles de )
CR .( │ Dans  ce   cas c'est  l'écran qui est une fenêtre
visible posée s)
CR
.( └───────────────────────────────────────)
WIN2 TAKE# 0 0 AT
.( ──────────────────────────────────────┐)
CR .(
partie droite  │)
CR .( l'écran en largeur comme en hauteur et être déplacée
partout.  │)
CR .( ur la fenêtre surdimensionnée  pour des applications
spéciales. │)
CR                                                    .(
──────────────────────────────────────────────────────
──────┘)
-64 0 AT# TAKE#
WIN1 CLOSE# 18 8 AT# OPEN# WIN2
::  19 0  DO I 3 * 64 - I AT# OPEN# TEMPO CLOSE# LOOP ;
OPEN#
26 3 WINDOW WIN3 0 0 AT
   .( ┌────────────────────────┐)
CR .( │ Appuyez sur une touche │)
CR .( └────────────────────────┘)
WIN3 TAKE# KEY DROP 15 ATTFILL#
DARK
   .( ╔════════════╗ )
CR .( ║ Fenêtre n°4 ║ )
CR .( ║ la position ║█)
CR .( ║ d'une fenê- ║█)
CR .( ║ tre peut    ║█)
CR .( ║ sortir de   ║█)
CR .( ║ l'écran.Les ║█)
CR .( ║ coordonnées ║█)
CR .( ║ peuvent     ║█)
CR .( ║ être néga-  ║█)
CR .( ║ tives. La   ║█)
CR .( ║ seule par-  ║█)
CR .( ║ tie visible ║█)
CR .( ║ est placée  ║█)
CR .( ║ sur l'écran ║█)
CR .( ╚════════════╝█)
CR .(  ██████████████)
```

```
16 17 WINDOW WIN4 WIN4 TAKE#
DARK
:: 30 0 DO I 0 AT# OPEN# CLOSE# LOOP ;
29 3 AT# OPEN#
WIN3 48 20 AT# OPEN# KEY DROP CLOSE#
WIN1 10  7 AT# OPEN#
WIN3 50 18 AT# OPEN# KEY DROP CLOSE#
WIN1 CLOSE# WIN4 CLOSE# WIN1 OPEN# WIN4 OPEN#
WIN3  0  4 AT# OPEN# KEY DROP CLOSE#
WIN2  5 14 AT# OPEN#
WIN3  1 12 AT# OPEN# KEY DROP CLOSE#
WIN1 OPEN#
WIN3 30 12 AT# OPEN# KEY DROP CLOSE#
WIN2 CLOSE#
WIN1 OPEN# CLOSE# 112 ATTFILL# 30 12 AT# OPEN#
WIN3 43 22 AT# OPEN# KEY DROP CLOSE#
WIN1 CLOSE#
40 11 WINDOW WIN5
WIN5
:: 8 0 DO 10 10 AT# TAKE# 10 9 AT# OPEN# TEMPO TEMPO LOOP ;
FORGET TEMPO
0 14 AT .( curieuses ces 2 fenêtres télescopées, non ? ) cr
ECHO ON
EOF \ Fin du listing d'essais
```

Une fenêtre est un objet très simple en FORTH. Il s'agit d'un espace affichable de l'écran vidéo qui se "superpose" à celui-ci quand on l'ouvre et qui le restitue quand on le ferme. Cet espace est mémorisé avec tous ses attributs d'affichage dans le corps (pfa) du mot désignant une fenêtre. La dimension d'une fenêtre en nombre de colonnes et de lignes peut dépasser celle de l'écran. Sa position est variable dans le plan étendu de l'écran. Seule la partie visible de la fenêtre entre les colonnes 0-79 et entre les lignes 0-24 de l'écran est affichée. La fenêtre possède un tampon d'écran conservant l'arrière-plan lors de son ouverture. Cette zone tampon est dimensionnée à la plus grande sous-fenêtre affichable selon les dimensions propres de la fenêtre complète.

L'utilisateur n'a besoin que de 5 mots pour gérer un environnement complexe de fenêtres affichables :

x y WINDOW <nom-de-fenêtre> définit une fenêtre de x colonnes et de y lignes. A l'exécution du <nom-de-fenêtre> cette fenêtre devient la fenêtre courante sur laquelle s'exercent les mots suivants pour la positionner, la remplir, l'ouvrir ou la fermer. Lors de sa définition la position d'une fenêtre est initialisée en 0,0 (coin haut gauche de l'écran) et son contenu est vide.

x y AT# positionne en colonne x et ligne y la fenêtre courante. Ces coordonnées peuvent sortir des limites physiques de l'écran: x et y peuvent être négatifs, x peut dépasser 79, y peut dépasser 24. Ces 2 derniers cas impliquent évidemment qu'aucune portion de la fenêtre ne sera affichée lors de son ouverture.

TAKE# (--) initialise le contenu d'affichage d'une fenêtre sur ce que contient l'écran à ce moment-là à la position définie de la fenêtre courante. Si les dimensions de la fenêtre dépassent celles de l'écran, il faudra plus d'une procédure TAKE# pour la remplir en variant sa position AT# en dehors de l'écran.

Pour des applications compilées comportant des fenêtres prédéfinies, utilisez en interprétation l'affichage par .(...) pour éditer la fenêtre puis chargez-la avec un ou plusieurs TAKE#.

OPEN# (--) ouvre la fenêtre courante à sa position courante.

CLOSE# (--) ferme la fenêtre courante en restituant l'écran tel qu'il était avant son ouverture.

Quelques précisions techniques pour développements ultérieurs tels que sauvegarde sur disque d'une fenêtre, modifications des attributs, édition directe dans une fenêtre, redirection de l'affichage dans le cadre restreint d'une fenêtre à l'écran:

Le PFA d'une fenêtre est ainsi organisé:

```
pfa   : largeur de la fenêtre (maximum 255 colonnes)
pfa+1 : hauteur de la fenêtre (maximum 255 lignes)
pfa+2 : position 1ère colonne sur 16 bits  signés  (0=1ère
          col. écran)
pfa+4 :  position 1ère ligne sur 16 bits signés (0=1ère
      ligne écran)
pfa+6 : contenu de la fenêtre qui occupe T1 octets.
        T1 = largeur * hauteur * 2  (avec les attributs)
pfa+T1: zone tampon d'écran qui occupe T2 octets
        (T2 <= T1)
        T2 = min(largeur,80) * min(hauteur,25) * 2
```

Dans chaque champ T1 et T2, le premier octet garde le caractère ASCII étendu, le deuxième garde l'attribut couleur ou monochrome dudit caractère.

Le mot TRANSFER (c1 seg1 adr1 c2 seg2 adr2 --) réalise toutes les opérations de transfert vers ou depuis l'écran. Il opère le déplacement de VISILIN lignes fois VISICOL colonnes depuis une zone située en seg1:adr1 organisée en c1 colonnes vers une zone seg2:adr2 organisée en c2 colonnes.

Sur certains systèmes, la répétition rapide de TRANSFER directement dans la mémoire vidéo peut provoquer des phénomènes de scintillement.

■

LA BIBLIOTHEQUE GEM DU VOLKSFORTH83

par F.I.G. HAMBURG (RFA)
Traduction B. THILLAYS

pour: VolksFORTH ATARI ST

Cette version du Volksforth83 contient une bibliothèque complète de routines GEM. Cette bibliothèque se compose de la partie AES ("Application Environment System") et VDI ("Virtual Device Interface"). On trouve aussi dans le volksFORTH la partie BASICS contenant la partie commune à AES et VDI.

Les noms des mots GEM disponibles correspondent aux routines C habituellement contenues dans les kits de développement. Nous avons néanmoins abandonné la plupart des préfixes pour économiser de la place en mémoire. Ces mots nécessitent les mêmes arguments, sauf ceux qui sont superflus. Certains programmes fournissent tellement de paramètres que mous avons préféré ne pas les mettre tous sur la pile. Dans ce cas il faut aller les chercher dans la table correspondante, ce que l'on trouvera dans la littérature appropriée. Les valeurs d'entrée restent sur la pile, (sous le nom de EVNT_MULTI).

Si vous êtes intéressé par la programmation GEM, consultez la documentation du kit de développement ATARI. Vous y trouverez (presque) toutes les fonctions (et beaucoup de fautes). La qualité n'est pas extraordinaire, en comparaison de celle pour IBM PC.

On peut aussi se réferer au manuel du compilateur C Megamax, bien qu'il soit également insuffisant.

Pour programmer sous GEM, un "Resource Construction Set" est nécessaire. C'est un programme qui permet la création d'icones, de menus déroulants, fenêtres d'alerte et de dialogue. Ce programme créé un fichier avec l'extension ".RSC" (fichier ressource) et un fichier ".H" permettant une liaison avec des programmes "C". Ce fichier contient donc les constantes relatives à chaque objet (un objet étant la plus petite subdivision d'un fichier "ressource"), qui doivent être compatibles avec votre programme Forth. Pour illustrer les différences, comparez le fichier EDIICON.H (C) avec EDIICON.SCR Un "Resource Construction Set" fait habituellement partie du kit de développement "C" (Megamax).

PRESENTATION DES MOTS GEM DU VOLKSFORTH

La bibliothèque GEM se compose de BASICS.SCR, VDI.SCR et AES.SCR. Nous présenterons les mots appartenant à ces fichiers, puis une bibliothèque de "super-mots" facilitant la manipulation des routines GEM.

LISTING PARTIEL DU FICHIER BASICS.SCR

```
Vocabulary GEM   GEM definitions also
\Le vocabulaire qui contient les mots GEM.

Create intin   $100 allot       Create ptsin   $100 allot
Create intout  $100 allot       Create ptsout  $100 allot
Create addrin  $100 allot       Create addrout $100 allot
\  Ces tables sont utilisées pour divers paramètres.
\  Leur taille peut varier, mais elle est  réduite
\  actuellement au minimum.

Variable grhandle
\ Cette variable stocke la donnée fournie par OPNVWK.

Create contrl   $16 allot
contrl 2 gemconstant opcode
       2 gemconstant #intin
       2 gemconstant #intout      ' #intout Alias #ptsout
       2 gemconstant #addrin
       2 gemconstant #addrout
       2 gemconstant function
\  Les composantes de cette table contiennent le nombre de
\  paramètres transmis par les tables de meme nom.

3 Create global   $1E allot
\  Cette table contient les valeurs décrivant
\  l'application.

addrout addrin intout intin global contrl 6 gemarray AESpb
        ptsout intout ptsin intin  contrl 5 gemarray VDIpb
\  Ces deux tables contiennent les pointeurs des tables
\  contenant les paramètres.....

Code array!   ( n0 ... nk-1 adr k --)
\ Stocke les k valeurs à partir de l'adresse adr dans
\ une table.
Code 4!         ( n1 .. n4 addr -- )
Code 4@         ( addr -- n1 .. n4 )
\ charge et lit les 4 valeurs à partie de adr. Utilisé
\ pour stocker et lire des rectangles dans les tables.

Code AES
( opcode #intin #intout #addrin #addrout -- intout@)
\   Gère tous les appels AES.
Code VDI       ( opcode #ptsin #intin --)
\   Gère tous les appels VDI.
: appl_init
\   Initialise une application, et doit être appelé à
\   chaque premier appel d'une fonction AES.
: appl_exit
\   Ferme une application.
Create sizes 8 allot
\   Cette table contient la taille de la matrice-caractères
\   en pixels.
: graf_handle
\   Charge la VDI-Handle dans la variable GRHANDLE et la
\   taille de la matrice-caractères dans SIZES
sizeconstant cwidth              sizeconstant cheight
sizeconstant bwidth              sizeconstant bheight
\   Noms des éléments des tables SIZES. Leur execution
\   fournit les adresses correspondantes.
: opnvwk
\   ouvre une "virtual workstation" et doit être executé
\   obligatoirement.
: clrwk
\   Efface la Workstation. Remplit l'écran avec la couleur
\   de fond.
: clsvwk
\   Ferme la "virtual workstation". Doit être appelé
\   juste avant la fin d'un programme.
: updwk
\   Met à jour la "virtual workstation". Attend la fin de
\   toutes les commandes VDI en cours.
: s_clip     ( x1 y1 x2 y2 clipflag -- )
\   Fixe la taille et la position du "Clipping rectangle"
\   (zone de capture souris) pour tous les appels VDI.
: grinit    appl_init graf_handle opnvwk ;
\   Effectue tous les appels nécéssaires pour ouvrir une
\   application.
: grexit    clsvwk appl_exit   ;
\   Idem pout la fin d'une application.
2Variable objc_tree
\   Contient l'adresse de l'arbre-objet, auquel se
\   rattachent les mots des librairies Menus Objet et
\   Forme. En  "C", il faut executer chacun de ces mots
\   independamment.
Variable c_flag
\   Dit si les appels de SHOW_C et  HIDE_C peuvent être
\   accumulés.
: show_c        ( -- )
\   montre la souris
: hide_c        ( -- )
\   cache la souris
```

LISTING PARTIEL DU FICHIER AES.SCR

```
Event :
: evnt_keybd    ( -- key )
\   Attend un évenement-clavier. key est constitué d'un
\   code de scrutation et d'une valeur ASCII.
: evnt_button  ( #clicks0 bmask bstate -- #clicks1 )
\    Attend  un évenement-touche. #clicks0 donne le compte
de \     CLICS,  bmask  et  bstate  spécifient  sur quelle
touche.
\   #clicks1 est le nombre de clics recus.
: evnt_mouse    ( f leftX topY width height -- )
\   Attend un mouvement de la souris. f=1 pour l'entrée et
\   f=0 pour la sortie dans le rectangle spécifié par
\   leftX topY width et height.
Create message $40 allot
\   Cette table est le buffer d'évenement-message.
: evnt_mesag    ( -- )
\   Attend un évenement-message.
: evnt_timer    ( dtime -- )
\   Attend un évenement  timer.  dtime  est  le  double du
temps
\   attendu en millisecondes.
Create events
\   Cette table contient les paramètres du mot EVNT_MULTI.
\   Le contenu de ce champ est le suivant:
\      +0 : évenements reconnus:
\           Bit 0     Clavier
\           Bit 1     Touche
\           Bit 2     Mouse rectangle 1
\           Bit 3     Mouse rectangle 2
\           Bit 4     Message
\           Bit 5     Timer
\                     +2    #clicks0
\                     +4    bmask
\                     +6    bstate
\                     +8    f leftX topY width height
\                    +18    ------- " -------------
\                    +28  dtime
: prepare
\   Ce mot copie le contenu de EVENTS dans la table de
\   paramètres correspondante. Il doit être appelé avant
\   EVNT_MULTI.
: evnt_multi     ( -- which )
\   Ce mot attend un des evènements possibles, spécifiés
\   dans la table EVENTS. "which" est le numéro de celui
\   qui est apparu, selon la même clef que le premier mot
\   d'EVENTS. Les paramètres de sortie doivent être
\   compatibles GEM...
: evnt_dclick  ( dnew dgetset -- dspeed )
\   Charge ou lit l'intervalle entre des clics multiples.
\   dgetset  =  1 signifie que dnew va être réinitialisé.
\   dspeed est l'intervalle mesuré.

Menu :

: menu_bar      ( showflag -- )
\   Efface ou affiche la barre des menus.
: menu_icheck   ( item showflag -- )
\   Autorise ou pas l'accrochage d'un des éléments du
\   menu.
: menu_ienable ( item enableflag -- )
\   Sélectionne ou non un des éléments du menu.
\   Un élément non séléctionné apparait en gris.
```

```
: menu_tnormal ( title normalflag -- )
\  Affiche le titre du menu en video inverse ou en normal.
: menu_text    ( item laddr -- )
\  Change le texte d'un des éléments du menu.
\  Sa longueur ne doit pas changer !
: menu_register( apid laddr -- menuid )
\  Installe un accessoire Desktop dans la liste des menus.
\  "menuid" est la position dans le menu.
```

Object :

```
: objc_add     ( parent child -- )
\  Assemble un objet enfant(child) à la fin de la liste
\  enfant de l'objet parent.
: objc_delete  ( object --  )
\  Efface un objet de l'arbre.
: objc_draw    ( startob depth x y width height -- )
\  Redessine l'arbre partant de startob et de profondeur
\  depth. x y width et height donnent un
\  Clipping-Rectangle.
: objc_find    ( startob depth x y -- obnum )
\  Cherche un objet à la position souris x y. Donne le
\  numéro de l'objet ou -1.
: objc_offset  ( object -- x y )
\  Donne la position écran d'un objet.
: objc_order   ( object newpos -- )
\  Déplace un objet à l'interieur de l'arbre.
: objc_edit    ( object char index kind -- newindex )
\  Utilisé pour éditer un texte à l'interieur d'un objet.
: objc_change
  ( object x y width height newstate redraw -- )
\  Modifie le status objet et redessine cet objet.
```

Form :

```
: form_do      ( ltree startobj -- objectno )
\  Cette routine pointe un objet et permet à
\  l'utilisateur de le modifier. objectno est le numéro
\  de  l'"exit-button".
: form_dial    ( diflag lix liy liw lih  bix biy biw bih )
\  Cette routine se compose de 4 routines, et sa fonction
\  est dictée par diflag:
\             0    Reserve une zone écran pour un objet.
\             1    Dessine une boite grandissante.
\             2    Dessine une boite rêtrecissante.
\             3    Libère une zone écran réservée.
: form_alert   ( defbttn 0string -- exbttn )
\  Permet de construire des "boites d'alerte" de façon
\  simple. defbttn est le bouton par défaut, 0string est
\  la chaine terminée par 0 qui caractérise la boite et
\   exbttn est le bouton activé.
: form_error   ( enum -- exbttn )
\  Dessine une boite d'alerte avec le texte:
\  "TOS-Fehler-Nummer ...." (erreur-TOS numéro ....)
: form_center  ( ltree -- x y width height )
\  Calcule la position de l'objet centré au milieu de
\  l'écran.
```

Graphic :

```
: graf_dragbox
   (  startx  starty width height  boundx  boundy
                  boundw boundh -- finishx finishy )
\  Dessine le contour de la boite qui peut être déplacée
\  par la souris sur l'écran. Le déplacement de la souris
\  est restreint à un rectangle externe.
: graf_movebox
  ( sourcex sourcey width height destx desty --)
\  Dessine une boite se déplaçant de source vers dest.
: graf_growbox     ( stx sty stw sth  fix fiy fiw fih -- )
\  Dessine une boite grandissante.
: graf_shrinkbox    ( fix fiy fiw fih  stx sty stw sth -- )
\  Dessine une boite rétrecissante.
: graf_watchbox
  ( object instate outstate -- inside/outside )
\  Controle le déplacement de la souris à l'interieur
\  d'un objet...
: graf_slidebox     ( parent object vhflag -- vhpos )
\  Controle le glissement d'une petite boite sur une
\  grande avec la souris...
2Variable mofaddr
\  Pointeur de forme-souris personnalisée.
: graf_mouse          ( mouseform -- )
\    Etablit une forme-souris :
\          0    Fleche
\          1    Curseur
\          2    Abeille
\          3    Doigt-pointeur
\          4    Main
\          5    Croix fine
\          6    Croix épaisse
\          7    Croix avec contour
\         255   Personnalisée
\         256   souris invisible
\         257   souris visible
: graf_mkstate       ( -- )
\  Charge la position de la souris et des boutons dans
\  les différentes tables.
```

Fileselect :

```
Create inpath
\  Table contenant le PATH par défaut, terminé par un
\  octet nul.
Create insel
\  Contient le nom du fichier sélectionné.
: fsel_input         ( -- button )
\  Dessine la boite de sélection de fichiers et attend le
\  choix de l'un d'eux. button est le bouton activé
\  (0 = ANNULE)
```

Window :

```
: wind_create
  ( components leftX topY maxWIDTH maxHEIGTH  --
   handle)
\  Ce mot met en place une fenêtre de taille maximum
\  avec tous ses éléments. Délivre un Handle propre à la
\  fenêtre.
: wind_open     ( W-handle leftX topY width height -- )
\  Dessine une fenêtre de taille spécifiée.
: wind_close    ( Whandle -- )
\  Ferme puis efface une fenêtre. Elle peut être ensuite
\  réouverte.
: wind_delete  ( Whandle -- )
\  Efface définitivement une fenêtre.
: wind_get      ( Whandle funktion# -- )
\  Donne toutes les informations sur une fenêtre...
: wind_set      ( Whandle funktion# par0 par1 par2  par3 --
)
\  Met en place les attributs d'une fenêtre comme la
\  ligne de titre, l'ascenseur etc.
: wind_find     ( mouseX mouseY -- Whandle )
\  Recherche le handle de la fenêtre située sous la
\  souris.
: wind_update  ( funktion# -- )
\  Utilisé pour mettre à jour l'affichage lors de la
\  manipulation de fenêtres ou de menus.
: wind_calc
  ( 0/1 components leftX topY width height -- )
\  Convertit les dimensions interieures d'une fenêtre en
\  extérieures(0) ou l'inverse (1). Le résultat est mis
\  dans les tables GEM.
```

RSRC :

```
: rsrc_load     ( addr -- )
\  needs address of 0-terminated $
\   Charge  un fichier  ressource en mémoire. addr pointe
une
\  chaine terminée par 0.
: rsrc_load"
\  Charge le fichier ressource dont le nom (terminé par
\  un  ") suit ce mot.
: rsrc_free     ( -- )
\  Libere une zone mémoire selon un fichier ressource.
: rsrc_gaddr    ( type index -- laddr )
\    Fournit  l'adresse  d'un  objet  dans   le  fichier
ressource.
: rsrc_saddr    ( type index laddr --)
\  Stocke en mémoire l'adresse d'un objet.
: rsrc_obfix    ( index laddr --)
\  Convertit un objet et la taille des caractères en un
\  ensemble de pixels.
```

LISTING PARTIEL DU FICHIER VDI.SCR

Fonction sortie

```
: pline        ( x1 y1 x2 y2 ... xn yn count -- )
\    Dessine une ligne brisée de x1,y1 à x2,y2 etc...
: pmarker      ( x1 y1 x2 y2 ... xn yn count -- )
\    Donne un Polymarker.
: gtext        ( addr count x y -- )
\    Affiche le texte aux coordonnées x y.
: fillarea     ( x1 y1 x2 y2 ... xn yn count -- )
\    Dessine un polygone coloré.
: contourfill  ( color x y -- )
\    Remplit la zone où se trouve x y.
: r_recfl      ( x1 y1 x2 y2 -- )
\    Dessine un rectangle sans bordure.
: GDP          ( #ptsin #intin functionno -- )
\    Routine générale de dessin.
: bar          ( x1 y1 x2 y2 -- )
\    Dessine un rectangle plein avec bordure.
: arc          ( angle-départ angle-fin x y rayon -- )
\    Dessine un arc de cercle.
: pie          ( angle-départ angle-fin x y rayon -- )
\    Dessine une part de camembert (ou de tarte).
: circle       ( x y radius -- )
\    Dessine un cercle.
: ellarc
  ( angle-départ angle-fin x y rayon-x rayon-y -- )
\    Dessine un arc d'ellipse.
: ellpie
  ( angle-départ angle-fin x y rayon-x rayon-y -- )
\    Idem mais plein.
: ellipse      ( x y xradius yradius -- )
\    Ellipse entière.
: rbox         ( x1 y1 x2 y2 -- )
\    Rectangle à bords arrondis.
: rfbox        ( x1 y1 x2 y2 -- )
\    Idem mais plein.
: justified    ( string x y length wordspace charspace -- )
\    Affiche un texte justifié sur une longueur définie.
```

Fonction Attribut :

```
: swr_mode     ( mode -- )
\    Définit le mode de sortie...
1 Setmode replace              2 Setmode transparent
3 Setmode exor                 4 Setmode revtransparent

: sl_type      ( style -- )
\    Définit le type des lignes...
1 Settype solid                2 Settype longdash
3 Settype dot                  4 Settype dashdot
5 Settype dash                 6 Settype dashdotdot
7 Settype userdef

: sl_udsty     ( pattern -- )
\    Définit le type des lignes personnalisées.
: sl_width     ( width -- )
\    Definit l'épaisseur des lignes.
: sl_color     ( color -- )
\    Définit l'index couleur des lignes.
: sl_ends      ( begstyle endstyle -- )
\    Définit le style de fin de ligne.
: sm_type      ( symbol -- )
\    Définit le type de polymarker

1 Setmtype point               2 Setmtype plus
3 Setmtype astérisque          4 Setmtype carré
5 Setmtype croix               6 Setmtype diamant

: sm_height         ( height -- )
\    Définit la hauteur des polymarkers
: sm_color          ( color -- )
\    Définit l'index couleur des polymarkers
: st_height         ( height -- )
\    Définit la hauteur des caractères (absolue)
: st_point          ( point -- )
\    Définit la hauteur des caractères (points)
: st_rotation       ( angle -- )
\    Définit l'angle de rotation de la ligne de caractères
: st_font           ( font -- )
\    Définit la police de caractères.
: st_color          ( color -- )
\    Définit la couleur du texte
: st_effects        ( effect -- )
\    Définit le style de l'écriture (gras, italique...).
: st_alignement     ( horin vertin -- )
\    Définit l'alignement des caractères
: sf_interior       ( style -- )
\    Définit le style de remplissage intérieur
: sf_style          ( styleindex -- )
\    Définit l'index de style de remplissage
: sf_color          ( color -- )
\    Définit l'index de couleur de remplissage de
polygones
: sf_perimeter      ( pervis -- )
\    Commute la fonction contours.
```

Operations "RASTER" (déplacements de blocs):

Les opérations "raster" servent à déplacer des zones écran,
aussi bien sur l'écran lui-meme qu'entre celui-ci et la mémoire.
Il s'agit donc de routines très rapides et il faut les utiliser chaque fois que l'écran doit être remis à jour. Toutes les autres routines d'affichage - du moins celles qui sont nécessaires - sont plus lentes.

```
  Variable >memMFDB
  Create scrMFDB
```

Les blocs "Memory Form Definition" décrivent la constitution d'un bloc de pixels en mémoire ou sur l'écran. Pour pouvoir travailler avec plusieurs zones mémoire, >memMFDB possède un pointeur sur la zone courante.

```
: copyopaque   ( Xfr Yfr width height Xto Yto mode --)
\ Routine de base pour tous les déplacements.

: scr>mem      ( addr_of_memMFDB -- )
\  Mot de définition pour toutes les déplacements
\  écran->mémoire

: mem>scr      ( addr_of_memMFDB -- )
\  Mot de définition pour toutes les déplacements
\  mémoire->écran

: scr>scr      ( Xfr Yfr width height Xto Yto --)
\ Déplace un rectangle sur l'écran.

Create memMFDB1
\ Bloc mémoire pouvant sauvegarder l'écran courant.

: scr>mem1     ( Xleft Ytop Width Heigth -- )
\ Sauvegarde l'écran en mémoire.

: mem>scr1     ( Xleft Ytop Width Heigth -- )
\ Restitue l'écran mis en mémoire.

: r_trnfm      ( -- )
\  Calcule les coordonnées standard selon la configuration
\  de l'appareil et inversement.

: get_pixel    ( x y -- color flag )
\ Donne la couleur d'un pixel.  Le flag est à 1 quand
\ il y a un point.
```

Entrée :

```
: sin_mode     ( devtype mode -- )
\    Définit le mode d'entrée....
: sm_locater   ( x y -- status )
\    Position de la souris, saisie de la position de
\    départ le cas échéant.
: sm_valuator  ( val_in -- status )
\    Gestion des modifications ....
: sm_choice    ( -- status )
\    Test des touches de fonction.
: sm_string    ( addr max_len echomode x y -- status )
\    Saisie d'une chaine avec écho.
: sc_form      ( addr -- )
\    Sélectionne la forme de la souris.
: ex_time      ( tim_addr -- long_otim_addr )
\    Lance une nouvelle routine d'interruption-timer.
```

```
: q_mouse      ( -- x y status )
\   Fournit l'état de la souris.
: ex_butv      ( pusrcode -- long_psavcode )
\   Lance une routine d'interruption "touche-souris".
: ex_motv      ( pusrcode -- long_psavcode )
\   routine d'interruption des déplacements
\   de la souris...
: ex_curv      ( pusrcode -- long_psavcode )
\   Routine d'interruption de gestion de forme
\   de la souris...
: q_key_s      ( -- status )
\   Fournit l'état des touches Shift.
```

Inquire :

```
: q_extnd            ( info_flag -- )
\   Cette fonction et les suivantes servent à établir
\   quelle fonction-VDI (ci-dessus) vient d'être appelée.
\   On ne peut pas les décrire rapidement, mieux vaut se
\   référer à la litterature spécialisée.
: q_color            ( color_index info_flag )
: ql_attributes      ( -- )
: qm_attributes      ( -- )
: qf_attributes      ( -- )
: qt_attributes      ( -- )
: qt_extent          ( string -- )
: qt_width           ( char -- status )
: qt_name            ( element_num -- )
: q_cellarray        ( cols rows x1 y1 x2 y2 -- )
: qin_mode           ( dev_type -- mode )
: qt_fontinfo        ( -- )
```

Escape :

```
: q_chcells          ( -- rows cols )
\   Taille de l'écran.
: exit_cur           ( -- )
\   Efface le curseur texte.
: enter_cur          ( -- )
\   Remet le curseur
: curup              ( -- )
\   vers le haut
: curdown            ( -- )
\   vers le bas
: curright           ( -- )
\   à droite
: curleft            ( -- )
\   à gauche
: curhome            ( -- )
\   curseur dans le haut à gauche
: eeos               ( -- )
\   efface jusqu'à la fin de l'écran
: eeol               ( -- )
\   ----- " ------------ de la ligne
: s_curaddress       ( row col -- )
\   positionne le curseur
: curtext            ( addr count -- )
\   affiche le texte
: rvon               ( -- )
\   video inverse on
: rvoff              ( -- )
\   ---- " ------ off
: q_curaddress       ( -- row col )
\   fournit la position curseur
: q_tabstatus        ( -- status )
\   fournit l'état de la souris
: hardcopy           ( -- )
\
: dspcur             ( x y -- )
\   positionne la souris
: rmcur              ( -- )
\   efface la souris
```

Les instructions suivantes servent à l'utilisation de dispositifs externes. C'est une partie plutôt mal documentée. On trouve quelques précisions dans le kit de développement de Digital Research. On se demande si toutes ces instruction fonctionnent sur l'Atari. Les essais dans ce domaine n'ont pas été très concluants jusqu'à présent.

```
: form_adv           ( -- )
\   mise en page imprimante.
: output_window      ( x1 y1 x2 y2 -- )
\   Imprime une fenêtre.
: clear_disp_list    ( -- )
\   Interrompt une impression, comme Clear Workstation,
\   mais sans linefeed.
: bit_image
 ( string aspect scaling num_pts x1 y1 x2 y2 -- )
\   Imprime un fichier 'Bit Image File'.
```

Les instructions suivantes sont tirées d'un "Output-driver" additionnel. Leur effet sur l'Atari reste inconnu. Elles ne semblent qu'à moitié executées...

```
: s_palette          ( palette -- selected )
\   Définit la palette de couleurs d'un moniteur IBM ??!!
: qp_films           ( -- )
: qp_state           ( -- )
: sp_state           ( addr -- )
: sp_save            ( -- )
: sp_message         ( -- )
: qp_error           ( -- )
: meta_extents       ( x1 y1 x2 y2 -- )
: write_meta         ( intin num_intin ptsin num_ptsin -- )
: m_filename         ( string -- )
```

■

INTERFACE FICHIERS DU VOLKSFORTH83

par F.I.G. HAMBURG (RFA)
Traduction: B. THILLAYS

Pour: VolksFORTH 83-Standard ATARI ST

INTRODUCTION

---> Création d'un fichier Forth:

1) Créer un fichier sur le disque:

```
MAKEFILE <nom>.scr
```

2) Spécifier sa longueur en nombre d'écrans:

```
n  MORE
```

le fichier contient maintenant les écrans 0 à n-1. Cette longueur peut être rallongée par d'autres MORE, mais pas diminuée.

----> Rappel d'un fichier du disque

Si le forth ne connait pas encore le fichier, il enverra le message "Haeh?". Entrer dans ce cas:

```
USE <nom>.scr
```

L'édition de l'écran n se fait alors par n L. Le mot USE crée un nouveau mot dans le dictionnaire nommé <nom>.scr, s'il n'existe pas encore. Pour voir si un mot existe déjà, utilisez WORDS ou simplement USE. Pour rappeler ensuite un fichier ainsi créé, il suffit de taper son nom...

Le fichier est créé dans le répertoire courant et sur l'unité de disque en place.

----> Sélection du répertoire

Il existe plusieurs méthodes. La plus simple pour rechercher un fichier dans le répertoire nommé test.dir est "dir":

```
dir  test.dir
```

test.dir est devenu le répertoire courant. S'il n'existait pas auparavant, on peut le créer par:

```
makedir  test.dir
```

GENERALITES

L'interface fichiers du volksforth utilise les fichiers du GEM-DOS et le système de sous-répertoires. Les fichiers Forth sont organisés de la façon habituelle en écrans de 1Ko. Les fichiers non conçus de cette façon sont aussi traités par blocs de cette taille. Par convention, les fichiers contenant des écrans Forth ont le suffixe .SCR . Les fichiers de données non ASCII ont le suffixe .BLK .

La commutation de l'interpretation fichier au mode direct se fait par:

DIRECT (--)
Le retour à l'interpretation fichier se fait par le nom du fichier.

DOS (--)
est le vocabulaire qui contient les noms des fichiers peu utilisés.

1) Le mécanisme des répertoires

La gestion des répertoires rappelle un peu celle de l'interpreteur de commandes du GEM-DOS: COMMAND.PRG.

PATH (--)
Ce mot permet, comme dans MS-DOS, d'indiquer au système la liste des répertoires dans lesquels le nom du fichier doit être recherché.

```
PATH dir1;dir2;....
```

<dir1>, <dir2> sont les chemins empruntés au cours de la recherche. Le système termine la recherche par le répertoire courant. Cette liste ne doit pas contenir d'espaces.

exemple: PATH A:\;B:\COPYALL.DEM;\AUTO\

Après avoir cherché sur la disquette A:, le système cherchera dans \COPYALL.DEM puis dans \AUTO\. Les \ sont importants, le premier indique que AUTO est un sous-répertoire du répertoire principal, et le deuxième distingue les noms de chemins des noms de fichiers. On peut aussi entrer:

```
PATH ;
```

Dans ce cas, seul le répertoire courant sera scruté. Et pour finir:

```
PATH
```

affiche la liste des chemins actuellement scrutés.

MAKEDIR cccc (--)
créé un sous-répertoire de nom CCCC dans le répertoire courant. Equivalent de MD dans l'interpreteur de commandes.

A: (--)
Sélectionne le lecteur A comme unité courante. (voir SETDRIVE). Idem pour B: C: D: .

SETDRIVE (n --)
Sélectionne le lecteur de numéro n comme unité courante. n=0 signifie A, n=1 signifie B etc.

DIR cccc (--)
Spécifie CCCC comme répertoire courant. Tous les nouveaux fichiers seront créés dans ce repertoire. Si CCCC n'est pas entré, DIR affiche le nom entier du répertoire. Equivalent de CD dans l'interpreteur de commandes.

FILES (--)
Liste le contenu actuel du sous-répertoire courant à l'écran. Les sous répertoires sont précédés d'un D. Equivalent de la commande DIR de l'interpreteur de commandes.

FILES" cccc" (--)
Liste les fichiers dont le nom est CCCC. CCCC admet les jokers ou les noms de sous-répertoires. Si aucun chemin n'est précisé, l'actuel est pris par défaut.

SAVESYSTEM <nom>
Permet de stocker le système sous le nom <nom>. Ce mot permet surtout de créer une application. Dans ce cas, il faut patcher le mot qui lance l'application dans 'COLD. Exemple: vous avez écrit un programme de copie, le mot principal s'appelle COPYDISK. En entrant la séquence:

```
' copydisk is 'cold
savesystem  copy.prg
```

vous créez un programme nommé COPY.PRG qui exécute automatiquement COPYDISK.

FICHIERS

Les fichiers sont constitués d'un nom FORTH et d'un nom GEM.DOS, differents l'un de l'autre. Pour exprimer un mot forth relatif à un fichier, nous parlerons d'un "Forthfile" dans ce qui suit. Pour un fichier présent sur la disquette, nous parlerons d'un GEM-file. Taper le nom du Forthfile le rend fichier courant pour toutes les opérations comme LIST, CONVEY, etc... L'ouverture d'un fichier est effectuée par sa recherche dans la liste de chemins contenue dans PATH. Lorsque le fichier est ouvert, cette liste de chemins peut être modifiée à volonté, sans changer les références du fichier. Pour des raisons de sécurité, il vaut mieux garder les fichiers fermés le plus souvent possible. La modification du PATH peut alors empêcher le système de trouver le fichier.

FILE <nom> (--)
Créé un mot forth portant le nom <nom>. Si <nom> est ensuite exécuté, il sera considéré comme fichier courant. Il sera aussi considéré comme FROMFILE par CONVEY. La correspondance entre un Forthfile et un GEM-file est créée par MAKE ou ASSIGN.

MAKE cccc (--)
Créé un GEM-file de nom CCCC dans le répertoire courant et établit sa correspondance avec le Forthfile actuel. Sa longueur est nulle et doit être fixée par MORE. Ex: FILE test.scr test.scr MAKE test.scr, créé un mot forth "test.scr" puis un fichier de même nom. Toutes les opérations comme LOAD, LIST etc. seront relatives à l'écran donné de TEST.SCR. Notez bien que le fichier est vide et que des opérations de saisie genèrent un message d'erreur.

MAKEFILE <nom> (--)
Créé un Forthfile de nom <NOM> puis un GEM-file de même nom. La séquence suivante est équivalente:

```
FILE <nom> <nom> MAKE <nom>
```

ASSIGN cccc (--)
Assigne le fichier courant au GEM-file de nom CCCC. Un message d'erreur apparait si le fichier est introuvable.

USE <nom> (--)
C'est le mot le plus important pour sélectionner un fichier. Rend courant le fichier nommé <NOM>. S'il est introuvable dans les répertoires de PATH, il est créé dans le répertoire courant, le message "FILE NOT FOUND" est émis et le Forthfile est ajouté au dictionnaire (à effacer eventuellement par FORGET).

CLOSE (--)
Ferme le fichier courant. La disquette est mise à jour et le Handle correspondant est libéré. Tous les blocs de ce fichier qui sont en mémoire sont sauvegardés puis effacés de la mémoire.

OPEN (--)
Ouvre le fichier courant. Un message d'erreur apparait si le fichier est introuvable. L'utilisation de ce mot est le plus souvent superflue, car cette opération est effectuée automatiquement lors d'un chargement.

FROM <nom> (--)
<NOM> est le nom du Forthfile qui va être copié par CONVEY ou COPY. Exemple:

```
filea 1 FROM fileb 3 copy
```

copie le bloc 1 de fileb dans le bloc 3 de filea. Ce mot utilise USE pour sélectionner le fichier. Cela signifie que fileb sera automatiquement créé s'il est introuvable.

LOADFROM <nom> (n --)
Charge le bloc n du Forthfile <nom>. Ce mot est pratique pour mixer le chargement de plusieurs fichiers. On peut ainsi imiter les fonctions d'un linker. Ce mot utilise également USE. Malgré les apparences, ce mot n'a rien à voir avec FROM ou FROMFILE !

INCLUDE <nom> (--)
Charge complètement le Forthfile <nom>. L'écran 1 doit contenir les instructions de chargement des autres écrans. Voir LOADFROM.

CAPACITY (-- u)
u est la longueur du fichier courant. Notez que cette longueur est supérieure de 1 au numéro du dernier bloc, car les blocs sont numérotés à partir de 0.

FORTHFILES (--)
Affiche la liste des Forthfiles, de leurs GEM-files correspondantsleur longueur et leur Handle.

FROMFILE (-- adr)
Adr est l'adresse de la variable qui pointe sur le Forthfile, qui est lue par CONVEY et COPY. Voir FROM. Cette variable est mise à jour par l'expression d'un Forthfile.

FILE? (--)
Affiche le nom du Forthfile courant.

MORE (n --)
Allonge le fichier courant de n écrans. Ces écrans sont rajoutés à la fin du fichier. Celui ci est ensuite fermé.

(MORE (n --)
Même chose que MORE, mais le fichier n'est pas fermé.

EOF (-- f)
f est un flag, vrai si la fin du fichier vient d'être lue. f est faux si le dernier octet lu n'est pas le dernier du fichier.

NOTES

L'effacement d'un Forthfile par FORGET, EMPTY etc.. efface automatiquement les tampons, après les avoir sauvegardés s'ils ont été modifiés. Le fichier est ensuite fermé.

L'utilisation de FLUSH ferme tous les fichiers. FLUSH doit être exécuté avant tout changement de disquette, non seulement pour mettre à jour la disquette, mais pour fermer les fichiers. S'il existe des fichiers de même nom sur la nouvelle disquette, les différences de longueur des fichiers ne seront pas comprises par Forth. L'utilisation de VIEW permet d'ouvrir automatiquement le bon fichier.

4) DIVERS

Le fichier DIVERSES.SCR contient quelques mots difficiles à classer.

>absaddr (addr -- abs_laddr)

convertit une adresse relative du Forth-system en une adresse absolue sur 32 bits.

.blk (--)
Affiche le numéro du bloc en train d'être compilé. La compilation de l'écran 1 donne aussi le nom du fichier.

abort((f --)
utilisé de la façon suivante: ABORT(chaine). Si f est vrai, la chaine est affichée. Comparable à ABORT" mais à utiliser en mode direct.

arguments (n --)
Vérifie qu'au moins n valeurs soient présentes sur la pile. Si ce n'est pas le cas, interrompt l'exécution avec un message d'erreur.

cpush (addr len --)
comme PUSH, mais pour une zone mémoire qui doit être restaurée apres l'exécution d'un mot. Permet la création de "tables locales".

bell (--)
envoie un "bip" sur la console.

blank (addr len --)
Remplit la zone spécifiée avec des espaces

setvec (--)
Positionne un "Critical error handler" du système d'exploitation sur une routine qui affiche une boite d'erreur d'accès disque. Les autres boites qui peuvent tourner sous ce Handler sont par exemple une invitation à changer de disquette...

restvec (--)
Restaure un "Critical error handler" à son ancienne valeur. Cette routine doit être exécutée avant de quitter Forth, sous peine de planter le système.

■

DOCUMENTATION DU SIMULATEUR DE CPU 1802 RCA

par G. DUMUR

pour : TURBO-Forth
diffusion : téléchargement 3615 SAM*JEDI
et module M4 TURBO-Forth

Objet du programme: simuler les fonctionnalités logicielles d'un cpu afin d'exécuter des programmes chargés sous forme binaire (code-machine); la vitesse d'exécution n'est pas conservée mais les possibilités de mise au point sont les mêmes que sur un outil de mise au point classique.

Le logiciel a été écrit essentiellement pour tester la faisabilité d'un simulateur en Forth avec comme projet le développement d'un simulateur spécialisé pour DSP Texas série 320. Aucune difficulté n'a été rencontrée et le temps de mise au point a été extrêmement court. Cependant de nombreuses extensions sont envisageables et facilement réalisables (gestion de la mémoire simulée en ROM/RAM, comptage du temps d'éxécution, utilisation d'un fichier d'événements pour simuler la marche en temps réel etc..).

De même une implantation pour une machine moins exotique (monochip en particulier) sera facile et utile.

IMPLANTATION:

Une zone de 64k de mémoire est réservée pour simuler la mémoire de la machine virtuelle (dans l'état actuel et faute d'information suffisante sur l'allocation mémoire de TURBO-Forth et surtout de l'éditeur aucune protection n'a pu être réalisée sur cette machine virtuelle, donc le contenu n'est pas forcément valide après une session d'édition): cette zone débute à l'adresse 0000h du segment calculé ainsi: (segment-courant-forth)+2000 en hexa.

Les registres du cpu sont simulés par des variables ou des tableaux.

MACHINE VIRTUELLE:

- un tableau de 16 registres 16 bits (R0-RF) subdivisables en moitié haute et moitié basse: accès 16 bits seulement en écriture:

```
val16b numéro-registre r !
```

en lecture:

numéro-registre r @ (ou r ?);

- un accumulateur 8 bits (D):

en écriture val 8b d! en lecture d@;

- un registre 1 bit de retenue(DF):

(e) val1b df! (1) df@;

- 2 registres 4 bits de désignation des pointeurs (X pointeur données et P pointeur programme) indiquant lequel des 16 registres joue le rôle précité:

accès (e) val4b x ! ou p !;(1) x @ p @;

- un registre T de sauvegarde des désignateurs (pas d'accès normalement pour le programmeur);

- une bascule 1 bit Q correspondant à un fil de sortie du cpu;

- 4 bits EF correspondant à l'état de 4 fils d'entrée du cpu EF1-EF4 positionnables individuellement par:

numéro sef(set) et numéro ref (reset);

- 7 registres entrée et 7 registres sortie correspondant aux 7 adresses d'entrée-sortie 8bits du cpu accès:

(e) val8b numéro inp! et val8b numéro out!;

COMMANDES: FONCTIONNEMENT DE LA MACHINE SIMULEE:

état-machine: visualise l'état courant de tous les registres avec ou sans les entrées sorties selon l'état de la bascule io;

io on ou off: positionne la bascule io pour l'affichage;

g: exécution du programme résident dans la machine virtuelle à partir de la position pointée par le registre pointeur programme courant; si le mode trace est activé, toutes les instructions exécutées sont affichées (en code hexa) sinon le système est muet; dans tous les cas, l'exécution sera interrompue soit par l'action sur une touche quelconque, soit lorsque le compteur programme correspond exactement à une adresse déposée sur la pile breakpoint;

tron et troff:positionne le mode trace;

0bk: vide la pile des points d'arrêt;

adresse +bk: empile 1 point d'arrêt;(16 points maximum)

-bk: dépile le dernier point d'arrêt;

?bk: visualisation de la pile des points d'arrêt;

NOTE: lorsqu'un point d'arrêt est atteint il ne peut être franchi qu'en pas-à-pas ou bien en le supprimant de la pile;

t: exécution d'une seule instruction (pas-à- pas);

GESTION DE LA MEMOIRE SIMULEE:

adresse sload nomfic.ext:
charge le fichier binaire nomfic.ext en adresse dans la machine simulée;

adresse-début adresse-fin ssave nomfic.ext:
sauve la zone depuis adresse-début jusqu'à (adresse-fin)-1 dans le fichier nomfic.ext;

adresse-début adresse-fin sdump:
visualise la zone mémoire;

adresse vedit: éditeur mémoire:
affiche l'adresse demandée et son contenu sera suivi de:

- espace: passe à l'adresse suivante;
- backspace (delete): passe à l'adresse précédente;
- cr (retour-chariot): quitte l'éditeur;
- caractère hexa: écrit la valeur qui sera formée par les 2 derniers caractères hexa à l'adresse courante;

Voilà, c'est tout pour l'instant.

```
( ************************** SIMULATEUR DE  CPU 1802 )
vocabulary sim02
only forth also sim02 also sim02 definitions
HEX
( **************************SIMULATION DE 64K DE MEMOIRE )
variable machine
( contient 0 si machine hote le segment courant si virt)
0 machine !
variable host variable virt
segment> @ dup host ! 2000 + virt !
( mots d'acces a la memoire du simule)
: vc@ (  adresse virtuel-c-fetch resultat)
   virt @ swap lc@ ;
: vc! virt @ swap lc! ;
( pas de version 16 bits necessaire pour le 1802)
: toggle-virt ( bascule en machine virtuelle)
  machine @ not
        if
        segment> @ machine !
        1000 segment> +!
        ( 64k reserves au dessus du segment forth)
        then cr ." VIRT." cr ;
: toggle-host  ( bascule en machine hote)
  machine @
        if machine @ segment> !
        0 machine ! ( drapeau faux)
        then cr ." HOST." cr ;

( ***** SIMULATION DES REGISTRES ET DRAPEAUX ***** )
: rtype create 20 allot does> swap 2* + ;
rtype r
: i/otype create 7 allot does> + ;
i/otype input i/otype output
variable DACCU  variable Q  variable EFI
variable p variable x variable rt variable CARRY
variable ie
: reset
  10 0 do i r 0 swap ! loop
  8 1 do i input 0 swap c! i output 0 swap c! loop
  0 p ! 0 x ! 0 rt ! 0 carry ! 0 daccu ! 0 q ! 0 efi ! ;
( ***** SIMULATEUR D'INSTRUCTIONS 1802 ***** )
( mots d'acces aux registres et mecanismes divers)
: d@ daccu @ ;      : d! daccu ! ;
: df@ carry @ ;     : df! carry ! ;
: sdf 0 df! ;       : rdf 1 df! ;
: 2p-1 1 swap 0 do 2* loop 2/ ;
: sef 2p-1 efi @ or efi ! ;
: ref 2p-1 -1 xor efi @ and efi ! ;
: inp! input c! ; : out! output c! ;
: pc p @ r ;
: pc@ p @ r @ ;
: pc! p @ r ! ;
: pc@@ p @ r @ vc@ ; ( m[r[p]] )
: pc@! p @ r @ vc! ;
: rx x @ r ;
: rx@ x @ r @ ;
: rx! x @ r ! ;
: rx@@ x @ r @ vc@ ; ( m[r[x]] )
: rx@! x @ r @ vc! ;
: pc+ p @ r +! ;
( instructions)
: ldn dup
   if ( si non 00=idle) 0f and r @ vc@ d! 1 pc+
   else drop then ;
: lda 0f and r  dup @ vc@ d! 1 swap +! 1 pc+ ;
: ldx  rx@@ d! 1 pc+ ;
: ldxa   rx dup @ vc@ d! 1 swap +! 1 pc+ ;
: str 0f and r @ d@ swap vc! 1 pc+ ;
: stxd  rx dup @ d@ swap vc! -1 swap +! 1 pc+ ;
: inc 0f and r 1 swap +! 1 pc+ ;
: dec 0f and r -1 swap +! 1 pc+ ;
: irx  rx 1 swap +! 1 pc+ ;
: glo 0f and r @ ff and d! 1 pc+ ;
: ghi 0f and r @ 100 / d! 1 pc+ ;
: plo 0f and r dup @ ff00 and d@ or swap ! 1 pc+ ;
```

```
: phi 0f and r dup @ ff and d@ 100 * or swap ! 1 pc+ ;
: inp 07 and input c@ dup rx@! d! 1 pc+ ;
: out 07 and output rx@@ swap ! 1 rx +! 1 pc+ ;
: skp  1 pc+ ;
: lskp  2 pc+ ;
: lsz  d@ 0= if 2 pc+ else 1 pc+ then ;
: lsnz  d@ if 2 pc+ else 1 pc+ then ;
: lsdf  df@ if 2 pc+ else 1 pc+ then ;
: lsnf  df@ not if 2 pc+ else 1 pc+ then ;
: lsq  q @ if 2 pc+ else 1 pc+ then ;
: lsnq  q @ not if 2 pc+ else 1 pc+ then ;
: lsie  ie @ if 2 pc+ else 1 pc+ then ;
: nop  1 pc+ ;
: sep 1 pc+ 0f and p ! ;
: sex 0f and x ! 1 pc+ ;
: seq  1 q ! 1 pc+ ;
: req  0 q ! 1 pc+ ;
: sav  rt @ rx@! 1 pc+ ;
: mark  x @ 10 * p @ + dup  rt !  2 r @ vc!   p @  x !
-1 2 r +! 1 pc+ ;
: ret
  1 pc+ rx@@ dup 0f and p ! 10 / x !   1 rx +!  1 ie ! ;
: dis
  1 pc+ rx@@ dup 0f and p ! 10 / x !   1 rx +!  0 ie ! ;
: shr  d@ 2 /mod d! df! 1 pc+ ;
: shrc  d@ 2 /mod df@ 80 * + d! df! 1 pc+ ;
: shl  d@ 2*  100 /mod df! d! 1 pc+ ;
: shlc  d@ 2* df@ + 100 /mod df! d! 1 pc+ ;
: add  rx@@ d@ + 100 /mod df! d! 1 pc+ ;
: adc  rx@@ d@ + df@ + 100 /mod df! d! 1 pc+ ;
: sd  rx@@ d@ - 100 /mod df! d! 1 pc+ ;
: sdb  rx@@ d@ - df@ 1 xor - 100 /mod df! d! 1 pc+ ;
: sm  d@ rx@@ - 100 /mod df! d! 1 pc+ ;
: smb  d@ rx@@ - df@ 1 xor - 100 /mod df! d! 1 pc+ ;
( instructions alu a 1 operande)
: ldi  1 pc+ pc@@ d! 1 pc+ ;
: ori  1 pc+ pc@@ d@ or d! 1 pc+ ;
: xri  1 pc+ pc@@ d@ xor d! 1 pc+ ;
: ani  1 pc+ pc@@ d@ and d! 1 pc+ ;
: adi  1 pc+ pc@@ d@ + 100 /mod df! d! 1 pc+ ;
: adci  1 pc+ pc@@ d@ + df@ + 100 /mod df! d! 1 pc+ ;
: sdi 1 pc+ pc@@ d@ - 100 /mod df! d! 1 pc+ ;
: sdbi 1 pc+ pc@@ d@ - df@ 1 xor - 100 /mod df! d! 1 pc+ ;
: smi 1 pc+ d@ pc@@ - 100 /mod df! d! 1 pc+ ;
: smbi 1 pc+ d@ pc@@ - df@ 1 xor - 100 /mod df! d! 1 pc+ ;
: short-branch 1 pc+
  0f and case
     0 of -1 endof ( br)
     1 of q @ 1 = endof ( bq)
     2 of d@ 0= endof ( bz)
     3 of df@ 1 = endof ( bdf)
     4 of 1 efi @ and 0= not endof ( b1)
     5 of 2 efi @ and 0= not endof ( b2)
     6 of 4 efi @ and 0= not endof ( b3)
     7 of 8 efi @ and 0= not endof ( b4)
     8 of -1 pc+ 0 endof ( skp )
     9 of q @ 0= endof ( bnq)
     a of d@ 0= not endof ( bnz)
     b of df@ 0=  endof ( bnf)
     c of 1 efi @ and 0= endof ( bn1)
     d of 2 efi @ and 0= endof ( bn2)
     e of 4 efi @ and 0= endof ( bn3)
     f of 8 efi @ and 0= endof ( bn4)
    endcase
  if pc@ ff00 and pc@@ or pc! else 1 pc+ then ;
: long-branch 1 pc+
  case
     c0 of -1 endof ( lbr)
     c2 of d@ 0= endof ( lbz)
     ca of d@ 0= not endof ( lbnz)
     c3 of df@ 1 = endof ( lbdf)
     cb of df@ 0= endof ( lbnf)
     c1 of q @ 1 = endof ( lbq)
     c9 of q @ 0= endof ( lbnq)
    endcase
  if pc@@ 100 * 1 pc+ pc@@ + pc! else 2 pc+ then ;
: $or 1 pc+ rx@@ d@ or d! ;
: $and 1 pc+ rx@@ d@ and d! ;
: $xor 1 pc+ rx@@ d@ and d! ;
( fin des instructions)
( **** DECODEUR D'INSTRUCTIONS ***** )
: decode ( adresse decode)
  pc@@ dup
```

```
  f0 and case
        00 of ldn endof
        10 of inc endof
        20 of dec endof
        30 of short-branch endof
        40 of lda endof
        50 of str endof
        80 of glo endof
        90 of ghi endof
        a0 of plo endof
        b0 of phi endof
        d0 of sep endof
        e0 of sex endof
        >r ( sauvegarde  pour equilibre pile)
        dup 61 67 between if dup inp then
        dup 69 6f between if dup out then
        dup c0 c3 between if dup long-branch then
        dup c9 cb between if dup long-branch then
        case
           f0 of ldx endof
           72 of ldxa endof
           f8 of ldi endof
           73 of stxd endof
           60 of irx endof
           f1 of $or endof
           f9 of ori endof
           f3 of $xor endof
           fb of xri endof
           f2 of $and endof
           fa of ani endof
           f6 of shr endof
           76 of shrc endof
           fe of shl endof
           7e of shlc endof
           f4 of add endof
           fc of adi endof
           74 of adc endof
           7c of adci endof
           f5 of sd endof
           fd of sdi endof
           75 of sdb endof
           7d of sdbi endof
           f7 of sm endof
           ff of smi endof
           77 of smb endof
           7f of smbi endof
           c4 of nop endof
           7b of seq endof
           7a of req endof
           78 of sav endof
           79 of mark endof
           70 of ret endof
           71 of dis endof
           c8 of lskp endof
           ce of lsz endof
           c6 of lsnz endof
           cf of lsdf endof
           c7 of lsnf endof
           cd of lsq endof
           c5 of lsnq endof
           cc of lsie endof
           endcase
     r>
     endcase ;
( fin du decodeur d'instructions)

( ****** MONITEUR DE SIMULATION ***** )
: 4type ( ecrit les 4 nibbles du nombre sur la pile)
  0 <# # # # # #> type ;
: 2type ( idem pour  2 nibbles )
  0 <# # # # # #> 2- swap 2+ swap type ;
variable io  io off
: ETAT-MACHINE ( affiche l'etat des registres internes)
  10 0 do i 8 mod 0= if cr then
       invers ." R" i . attoff space i r @ 4type space
       loop cr
 invers ." D: " attoff daccu @ space 2type 3 spaces
 invers ." DF " attoff carry @ 1 and space . 3 spaces
 invers ." P: " attoff p @ 0f and space . 3 spaces
 invers ." X: " attoff x @ 0f and space . 3 spaces
 invers ." T: " attoff rt @ space 2type 3 spaces
 invers ." IE " attoff ie @ space 2type 3 spaces
 invers ." Q: " attoff Q @ 1 and space . 3 spaces
```

```
invers ." EF " attoff efi @ 0f and 2 base !
space 4type hex cr
io @ if
8 1 do
   invers ." i" i . attoff
   space i input c@ 2type 3 spaces loop   cr
8 1 do
   invers ." o" i . attoff space i output c@
   2type 3 spaces loop    cr
then ;
( ****** EDITEUR DE MEMOIRE SIMULEE ***** )
variable adresse
: adr. adresse @ dup 10 mod 0= if cr dup 4type then
      space vc@ 2type ." -" ;
variable $key ( memoire tempo de key)
variable $temp ( accu de chiffres)
variable $num ( drapeau d'entree numerique encours)
: tonum ( code-ascii tonum fl result)
  dup 30 39 between
  if 0f and -1
     else dup 41 46 between
     if 37 - -1
        else 0
        then
  then ;
: vedit ( adresse vedit)
  0 $num ! cr dup adresse ! dup 4type space vc@ 2type ." -"
  begin key upc dup $key !
  $num @ if ( mode entree numerique en cours )
            $key @ tonum
            if $temp @ 10 * + $temp ! $key @ emit
               ( accumule les chiffres)
            else 0 $num ! ( quitte le mode numerique)
               $temp c@ adresse @ vc! ( mise en memoire)
            then
         else $key @ tonum
            if $temp ! -1 $num ! $key @ emit then
            ( memorise le 1er chiffre et passe )
            ( en mode numerique)
         then
         case
            20 of drop 1 adresse +! adr. endof
            08 of drop -1 adresse +! adr. endof
            2f of drop cr adresse @ dup 4type
                   space vc@ 2type ." -" endof
            0d of drop exit endof
            endcase
  again ;
( ****** COMMANDES PRINCIPALES ***** )
: pc. ( affiche le pc) pc@ 4type space pc@@ 2type ;
: 16table create 20 allot does> swap 2* + ;
  ( table de 16 entiers)
16table breaktable variable bkptr
  ( pointeur sur emplacement bk)
: 0bk 0 bkptr ! ;
( usage adresse +bk)
: +bk bkptr @ dup 10 =
   if drop ." 16 points d'arret maxi!" cr
   else breaktable ! 1 bkptr +! then ;
: -bk bkptr @ dup 0=
   if drop ." aucun point d'arret!" cr
   else -1 bkptr +! then ;
: ?bk bkptr @
   begin dup 0= not
   while
      1- dup breaktable @ 4type space
   repeat drop ;
: pcmatch
  bkptr @
  begin dup 0= not
  while  1- dup breaktable @ pc@ =
             if drop -1 exit then
  repeat  drop 0 ;
variable trace-simulateur
: tron trace-simulateur on ;
: troff trace-simulateur off ;
: g
  begin
  key? not pcmatch not and
  while trace-simulateur @
     if pc. cr then decode
  repeat
  etat-machine pc. cr ;
: t pc. decode etat-machine ;
: LLOAD ( seg adrdeb -lload--dr:nomfic.ext)
  ?OPEN DRV >R -1 ( 65535 octets a lire par defaut)
  EXT$ PAD PLACE " .COM"
  EXT$ $! FILENAME PAD COUNT EXT$ $! DUP 1+ C@
  ASCII : = IF DUP C@  ASCII A - SELECT THEN
 0 (OPEN) ?DOS-ERR DUP >R (GET) ?DOS-ERR . ." Octets lus"
cr
  R> (CLOSE) R> SELECT ;
: sload ( adrdeb sload nomfic)
  virt @ swap lload ;
: ssave ( adrdeb adrfin ssave nomfic)
  virt @ -rot lsave ;
: sdump
  virt @ -rot ldump ;
(  ***** FIN DU MONITEUR *****)
( FIN DU SIMULATEUR )
```

Die Fileselector-Box unter volksFORTH83 auf dem Atari ST

Bernd Pennemann, Hamburg

In dem folgenden Artikel wird die Handhabung der Fileselektor-Box, die das Betriebssystem des Atari ST zur Verfügung stellt, besprochen. Ein Beispiel demonstriert die Einbindung der Box in den Editor des volksFORTH83.

Schlüsselworte : Fileselectorbox, Atari ST, GEM, volksFORTH83

Die Box

In einem früheren Artikel wurden die Hilfsmittel, die das GEM (Graphics Environment Manager) auf dem Atari ST zur Verfügung stellt, erwähnt [1]. Zu den vordefinierten Objekten des GEM gehört die Fileselektor-Box. Das Bild zeigt, wie diese Box aussieht.
Sie dient der Auswahl von Files. Der Benutzer kann diese Box auf verschiedene Arten beeinflussen.
Er kann

-) durch Anklicken einen Filenamen aus dem Directoryfenster in das Auswahlfeld kopieren.

-) durch Klicken in den grauen Bereich des Sliders oder Ziehen des hellen Bereichs das Fenster über das Directory bewegen.

-) durch Anklicken eines Subdirectories oder des Schließsymbols das Indexfeld ändern.

-) durch Eingabe von der Tastatur aus sowohl das Indexfeld als auch das Auswahlfeld ändern. Dabei haben die Tasten <Esc> <Backspace> <Delete> <Return> sowie die Cursortasten eine besondere Funktion.

Die Box sollte in GEM-unterstützten Programmen verwendet werden. Sie hat den Vorteil, daß man leicht Subdirektories manipulieren kann. Außerdem wird dem Benutzer eine Liste der Wahlmöglichkeiten präsentiert. Die Auswahl der Laufwerke ist mit dieser Box allerdings sehr unbequem.
Um einen maximalen Bedienungskomfort zu erreichen, sollte die Box mit sinnvollen Vorbesetzungen der Felder erscheinen. In der Regel wird das Indexfeld das aktuelle Laufwerk, das aktuelle Subdirectory und ein "passendes" Wildcard enthalten (s.u.). Das Auswahlfeld kann z.B. das zuletzt ausgewählte File enthalten. In Applikationen, die mehrere Arten von Files mit Hilfe der Box auswählen, sollte für jede Art von Files eine eigene Vorbesetzung existieren, da Benutzer oft verschiedene Typen von Files in verschiedenen Subdirectories halten.

Beim Aufruf der Box werden zwei Strings mitgeliefert, einen für das Indexfeld und einen für das Auswahlfeld. Im Bild sind das die Strings "A:\GEM*.SCR" und der leere String gewesen. Nach Drücken von "OK" wird er Inhalt der beiden Felder in die Strings zurück übertragen, sie enthalten dann das ausgewählte File und Laufwerk sowie das Directory, in dem sich das File befindet. Um mit diesen Informationen das File öffnen zu können, müssen die Strings erst in für das GEMDos geeignete Stücke zerlegt werden. Der Indexfeld-String besteht aus

bis zu drei Teilen, nämlich :

[Laufwerk ":\"] [evtl: Subdirecories "\"] [Wildcard]

Das Wildcard bestimmt, welche Files des Directories überhaupt in dem Fenster präsentiert werden sollen. Im Bild sind das nur Files, die auf .SCR enden, also Files, die Screens für das volksFORTH83 enthalten. Dieser Wildcard wird nach Aufruf der Box nicht weiter benötigt und in unserem Programm einfach weggeschmissen. Laufwerk und Subdirectory werden mit Hilfe der Worte SETDIR und SETDRIVE an das GEMDos weitergereicht. Anschließend kann auf das File unter seinem Namen zugegriffen werden.

Das Programm

Das folgende Listing bindet die Fileselektor-Box in den Editor ein. Bei Aufruf des Wortes ED wird die Box gemalt, der Benutzer kann ein File auswählen und nach Anklicken von "OK" das File editieren. Es wird immer der Screen 1 des Files präsentiert. Das Wort ED ruft das Wort BOX_USE auf. BOX_USE entspricht in der Wirkung völlig dem Wort USE , daß im volksFORTH83 vorhanden ist. Der entscheidende Unterschied besteht darin, daß man bei BOX_USE nicht den Namen eintippen muß, sondern stattdessen mit Hilfe der Fileselector-Box ein File durch Anklicken auswählt. Diese beiden Worte sind zur Benutzung, d.h. zum Eintippen von der Tastatur, bestimmt. Die anderen Worte stellen die primitiven Funktionen dar, die für den Benutzer nicht so interessant sind.

Für eigene Applikationen, die nur ein File öffnen, daran herummanipulieren und es danach wieder schließen, ist das Wort GETFILE auf Screen 6 gedacht. Es hat den Vorteil, daß es kein Forthfile kompiliert. Die Erklärung des Unterschieds zwischen einem Dos- und einem Forthfile würde den Rahmen dieses Artikels sprengen; bitte schauen Sie in das Handbuch [2].

In den Worten OPEN_FILE und GETFILE wird der Suchpfad des Fileinterfaces (siehe PATH in [2]) mit PATHES OFF auf die Länge Null gesetzt. Damit der Pfad aber nicht verloren ist, wird seine Länge mit PATHES PUSH gerettet. Diese Maßnahmen sind erforderlich, damit der gerade ausgewählte File nur in dem ausgewählten Directory und nicht im gesamten Suchpfad gesucht wird. Andernfalls könnte es nämlich passieren, daß im Suchpfad ein File gleichen Namens aber in einem anderen Directory gefunden wird! Andererseits soll der Pfad nicht vollständig gelöscht werden, damit die VIEW-Funktion des Editors die zu suchenden Files auch finden kann.
Wie schon in [1] erwähnt, muß vor dem Aufruf von GEM-Funktionen, zu denen auch die Fileselektor-Box gehört, GRINIT ausgeführt werden. Soll die Applikation beendet werden, muß vorher ein GREXIT ausgeführt werden. Da der Editor auch eine GEM-Applikation darstellt, brauchen wir uns bei ED um dieses Problem nicht weiter zu kümmern. Im Wort GETFILE kann das jedoch erforderlich sein.

Eine Definition der Worte $ADD und $SUM findet man in [1] oder [2]. Hier wird noch eine zusätzliche Funktion implementiert, nämlich das Anfügen von einzelnen Zeichen an einen String. Dies geschieht mit Hilfe von ADDCHAR .

[1] B.Pennemann, Alert-Boxen unter volksFORTH83 auf dem Atari ST, Vierte Dimension II/4 (1986) S. 19-23

[2] B.Pennemann et.al., Handbuch für das volksFORTH83, 2.Auflage vom 18.12.1986, Selbstverlag (c) 1986, Hamburg

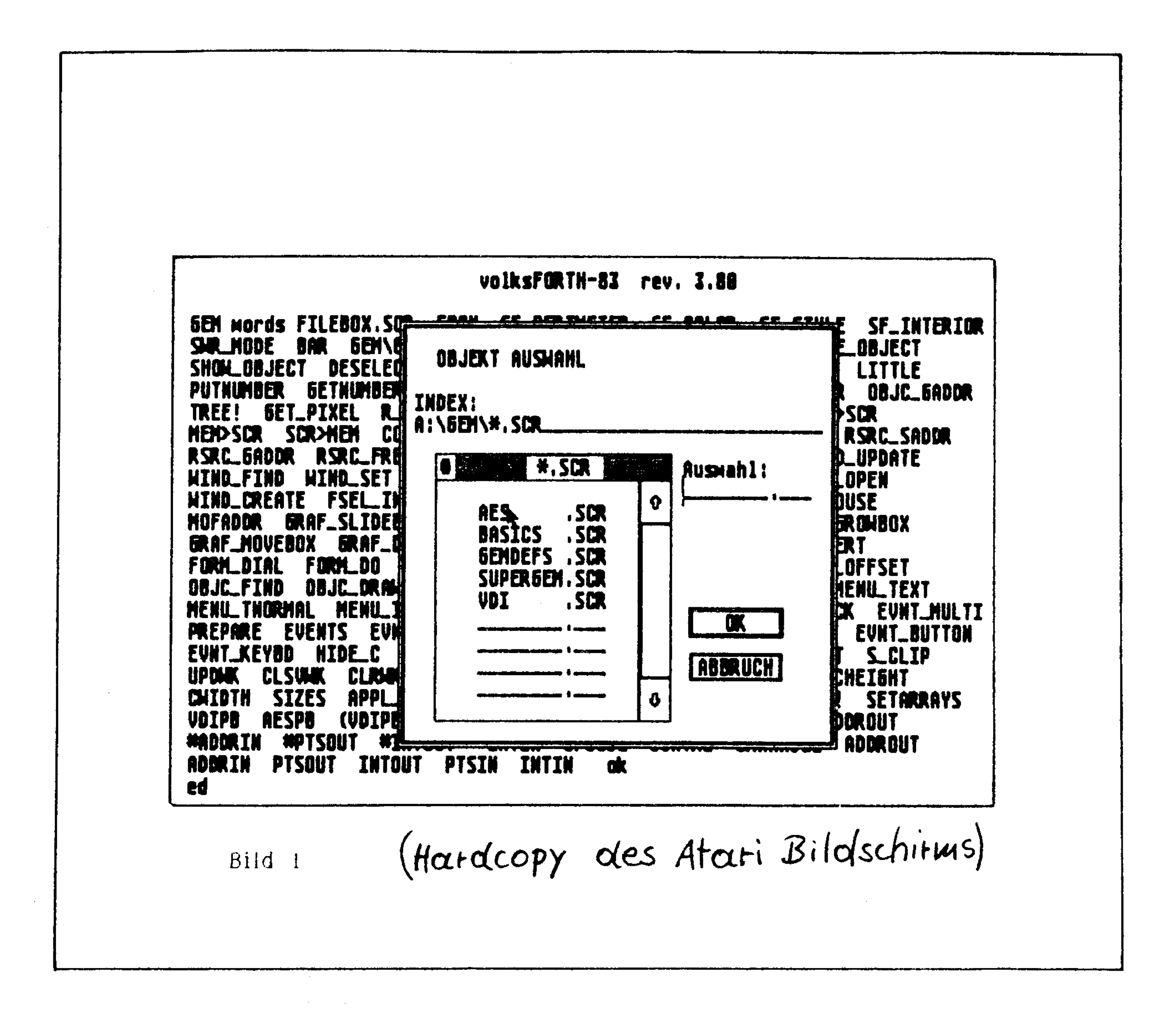


Bild 1 (Hardcopy des Atari Bildschirms)

Die Fehler

Die Fileselektor-Box des GEM hat natürlich auch Fehler. Mir sind zwei bekannt :

-) Beim Editieren des Indexfeldes durch Texteingabe und anschließendes Drücken von "OK" liefert die Routine den Wert für "CANCEL" statt für "OK". Man muß erst den Balken rechts von der Liste der Files anklicken.

-) Wenn man nach Editieren des Indexfeldes den grau schattierten Bereich oberhalb des Fensters anklickt, um das GEM zu veranlassen, die Files entsprechend dem neuen Wildcard zu präsentieren, wird das Wildcard auf "*.*" gesetzt.

0

```
          ## EDBOX.SCR ##                    bp 26dec86

Dieses File erweitert den Editor um eine Fileselectorbox, die
 nach Aufruf des Wortes ED erscheint.

Die Behandlung der Box sollte beispielhaft alle Eventualitäten
 abdecken und kann direkt in eigene Applikationen eingebaut
 werden.
```

7

1

```
\ Loadscreen                                  bp 26dec86

Onlyforth

\needs GEM                  include GEM\AES.SCR


\ Create charhold   1 allot

: addchar       ( c -- )        \ add char to  string
   charhold c!   charhold 1 $add  ;


1 4 +thru
```

8

bp 26dec86

Durchsuche nur FORTH und ONLY ; ich hätte auch Gem also Dos also schreiben können, aber durch explizites voranstellen (s.u.) der Vokabulare wird klar, wo bestimmte Worte zu finden sind...

ADDCHAR entspricht $ADD , jedoch für einzelne Zeichen

2

```
\ select the right files                      bp 26dec86

: getsubdir     ( -- adr len )  \ get actual subdirectory
   here   [ Dos ]  getdrive 1+  getdir ?diskabort
   here scan-name ;

: default        ( adr -- )      \ initialize  fileselectorbox
   [ Gem ] inpath off  inpath $sum !
   [ Dos ] getdrive Ascii A + addchar   Ascii : addchar
          getsubdir ?dup
          IF   $add  Ascii \ addchar  ELSE  drop  THEN
   count  $add  0 addchar ;
```

9

bp 17jan87

GETSUBDIR Das aktuelle Subdirectory wird durch Aufruf der GEMDOS-Funktion geholt und bei HERE abgelegt. Die Länge wird mit SCAN-NAME festgestellt.

DEFAULT Initialisiert die Fileselektorbox. Das 'Index'-Feld wird mit dem aktuellen Laufwerk, dem aktuellen Subdirectory und einem Wildcard-String, dessen Anfangsadresse auf dem Stack liegt, initialisiert. Der Inhalt des 'Index'-Feldes befindet sich ab INPATH im Speicher!

3

```
\ strip drive and path from INPATH            bp 26dec86

: strip_wildcard ( -- )          \ delete the file-wildcard
   [ Gem ] inpath count +
   BEGIN  dup c@  Ascii \ -
   WHILE  1-   REPEAT                1+ off ;

: set_disk      ( --- )          \ set drive and directory
   strip_wildcard (         ) inpath 1+
   dup 1+ c@ Ascii A
    IF dup c@ Ascii A  [ Dos ] setdrive  2+  THEN
   [ Dos ] (set    ) diskabort ;
```

10

bp 26dec86

STRIP_WILDCARD schneidet den Wildcard-String hinten wieder ab. Der Rest enthält dann nur noch Laufwerk und evtl. Subdirectory.

SET_DISK Wertet INPATH aus. Der Wildcard-String wird abgeschnitten, das evtl. vorhandene Laufwerk gesetzt und schließlich, falls vorhanden, das Subdirectory gesetzt.

4

```
\ open box and select a file                       bp 26dec86

: do_selector   ( -- f )        \ select a file, set direct.
  [ Gem ] fsel_input  1 -  dup ?exit  set_disk ;

: (use          ( adr len -- )  \ like USE , but takes a $
  dup #tib !  tib swap cmove  >in off
  use  ;

: open_file     ( -- f )        \ select a file for future use
  " *.SCR" default    do_selector    dup ?exit
  [ Dos ] pathes push  pathes off     \ equiv.  'PATH ;'
  [ Gem ] insel  count  (use  open ;
```

11

bp 26dec86

DO_SELECTOR malt die Fileselector-Box und wertet INPATH aus. Wird CANCEL gedrückt, so ist das gelieferte Flag TRUE !

(USE wie USE , jedoch wird der Name des zu benutzenden Files auf dem Stack erwartet. Da es keine primitive Form von USE gibt, muß der String erst in den Text Input Buffer (TIB) kopiert werden, von wo ihn USE einliest.

OPEN_FILE Selektiert mit Hilfe der Fileselector-Box Laufwerk, Subdirectory und File. Sucht und öffnet das File. BLOCK LIST V L etc. beziehen sich auf das File.

5

```
\ main word : display box, use choice              bp 17jan87

| : desktop       ( -- x y w h )  \ Größe des desktops
   0 0 [ Gem ] cwidth &80 *   cheight &25 * ;

| : savescreen    ( -- )  [ Gem ] desktop scr>mem ;
| : restorescreen ( -- )  [ Gem ] desktop mem>scr ;

: box_use       ( -- )          \ query a filename
  savescreen
      [ Gem ] show_c  open_file  hide_c
  restorescreen
  abort" dann nicht !" ;

: ed            ( -- )
  box_use  capacity 1-  1 min  l ;             \ edit screen 1
```

12

bp 1jan87

DESKTOP liefert Lage und Größe des Bildschirms in Pixeleinheiten. (Normalerweise 0 0 640 400)

SAVESCREEN kopiert den Bildschirminhalt in einen Buffer ..
RESTORESCREEN .. und wieder zurück.

BOX_USE Vor Aufruf der Box wird der Bildschirm gerettet, da das GEM nach Löschen der Box das von ihr belegte Rechteck in neutralem Atari-Grau ausfüllt. Wurde CANCEL gedrückt, so wird abgebrochen.

ED Selektiere ein File; rufe Editor mit Screen # 1 (falls er existiert) auf.

6

```
\ fileselector for Applications                    bp 26dec86

| file selected_file

: getfile       ( -- )          \ query a file
  selected_file  [ Dos ] close
  " *.SCR" default
  [ Gem ] show_c   do_selector  hide_c
          abort" dann nicht!"
  [ Dos ] pathes push  pathes off     \ equiv.  'PATH ;'
  [ Gem ] insel count 1+  isfile@
          [ Dos ] filename  swap   cmove
  open  ;
```

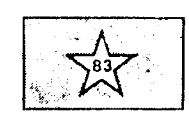
13

bp 26dec86

In Applikationen gibts nur ein Forthfile, nämlich dieses...

GETFILE ordnet mit Hilfe der Fileselectorbox dem Forthfile SELECTED_FILE ein Dosfile zu. Es wird auch gleich aufgemacht.

Classes in Forth



Vince D. Kimball
Ipswich, Massachusetts

If one wishes to do object-oriented programming in Forth, one must first add the class concept to the language. A Forth-like solution to the problem, a minor modification of the vocabulary concept, is proposed.

Overview

The principles of transparency and localization seem to be central to the current interest in object-oriented programming. Transparency emphasizes the wish to use generic operators across data structures, and localization emphasizes the desire to partition a group of data structures and operations upon them into a separate entity which may be understood more or less on its own. Currently, Forth does not seem to support these principles in any direct way. Multiple-code-field words are a first step toward generic operators, but they are flawed for general use in that they do not allow adding to the original class of operators to be used with a given data structure. They are useful, however, for the very basic operators which are common to most data structures. Vocabularies seem to provide localization, but at present they are insufficient to the task because they do not allow easy mixing of different vocabularies or the explicit specification of linkages among vocabularies.

If we accept these principles as useful but want to retain the flexibility and performance of Forth, we must discover how to add structures to Forth to support them without making Forth into a pale echo of Smalltalk. The proposed solution is to implement the class as a modified vocabulary and to enable the use of the class name as a prefix operator for modifying the dictionary search sequence. I believe that this unique concept will provide the power of object-oriented programming without sacrificing any of Forth.

Plan

An extremely simple method of adding classes to Forth involves the use of Forth's built-in vocabulary system as a foundation. The addition of six new words plus a modification of Forth's dictionary lookup sequence will provide the core of object programming while maintaining the idiom and flexibility of Forth. The first three new words **CLASS, CLASS@** and **<SUPER** allow for the definition of classes. The last three new words **CLASSVAR, DEFER** and **CLASS>** provide the useful ability to defer binding the name of a class to a word until run time. Other words may suggest themselves as more experience with this style of programming is gathered.

Class Definition Words

Classes would be defined according to the following form:

CLASS ClassName
<SUPER SuperClassName
CLASS@ ClassName **DEFINITIONS**
(definitions in class ClassName)
FORTH DEFINITIONS

The word **CLASS** would create (in the compilation vocabulary) a dictionary entry for ClassName which specifies a new list of word definitions forming the class being defined. Subsequent execution of ClassName will be as a prefix operator making the words in the class the first part of the search order during the next dictionary lookup. Thus, the phrase "ClassName WordName" would find the word WordName in the class ClassName, if there was one, and the search order would be the same after the phrase as it was before it. The word **<SUPER** would be used to indicate the superclass of the class just defined. It would chain the class indicated by ClassName to the class indicated by SuperClassName. When a dictionary search of ClassName is exhausted, SuperClassName would be searched. Those classes without superclasses could be declared as

CLASS ClassName **<SUPER** Object

The Object class would be the primary class, holding definitions common to all classes. Classes defined without using the **<SUPER** word would not be chained to any superclass, which might be useful in some cases. The word **CLASS@** would be used in the phrase "**CLASS@** ClassName" to make the following class name the first vocabulary in the regular search order, rather than the active class as it normally is.

Class Variables and Deferred Binding

As defined above, the class of an object must be known when the word involved is defined. In some cases it may be convenient not to have to specify the name of a class in advance. This ability is provided by employing the following phrase:
ClassVarName **DEFER** WordName

When this phrase is executed, WordName is looked up in the vocabulary in the class which is currently referenced by ClassVarName and then is executed. This lookup will take a certain amount of time, but the increase in flexibility may be worthwhile at times. It would be an error if WordName is undefined at run time, of course.

Class variables are defined by using the standard form:

CLASSVAR ClassVarName

This phrase would define a null class variable which would have to be assigned a real class to be of use. Unlike classes, class variables are not considered prefix operators because they execute at run time to provide information to **DEFER**. The method of assigning a class to a class variable had perhaps be best left to the discretion of the implementor, although the following form may be satisfactory:

CLASS> ClassName ClassVarName

The difficulty in implementing this operation is ensuring that ClassName is not executed as a prefix operator.

Dictionary Lookup

The final change to Forth to provide classes would be to modify its dictionary lookup sequence in order to enable the use of class names as prefix operators which modify the search order only on a temporary basis. The implementation of this new lookup sequence would seem to require that there be an **ACTIVECLASS** vocabulary to be searched before **CONTEXT** and **CURENT**. The execution of a class name would patch the **ACTIVECLASS** variable to allow searching the appropriate class. At the end of the search order, the **ACTIVECLASS** vocabulary would be set null. This implementation should not conflict with any other special vocabulary constructs, such as **ONLY**.

Application and Implementation

The general use for classes is to organize the dictionary according to the types of objects being used. For example, one could use the phrases "**SINGLE +**" to add single-length integers, "**DOUBLE +**" to add double-length integers and "**FLOAT +**" to add floating-point numbers if the classes **SINGLE**, **DOUBLE** and **FLOAT** had been defined to describe single-length integers, double-length integers and floating-point numbers respectively. Figure One lists the code for the same sample application which is used in the Smalltalk book. I have used the **ONLY** concept to avoid the necessity of writing **SINGLE** before each of the single-length operations, and I have left the implementation of an integer dictionary class to the readers. The example uses three operations from the IntDictionary class: (1) **at** to access the value corresponding to a certain code; (2) **isAt** to store a value corresponding to a certain code; and (3) **new** to create a new IntDictionary given the maximum number of codes involved. The Forth code and the usage examples should be relatively straightforward. However, it may be useful to point out that the words corresponding to the dictionary codes for income and expense categories are not defined in the example; these definitions are not essential to understanding the example and are of the form

codeValue **CONSTANT** codeName

Figure Two lists the code for implementing the words I have proposed under the Laxen/Perry F83 model. The code should be relatively straightforward, so I will only review some of the more challenging sections. The **CLASS** defining word produces a dictionary entry similar to that of the **VOCABULARY** defining word with the addition of space for a pointer to the class's superclass and a different run-time action. **DEFER** compiles the code address of its run-time word **(DEFER)** and a counted string representation of the word which follows it in the input stream. **(DEFER)** extracts the address of the string which follows it, moves the instruction pointer past the string, looks up the word in the dictionary and either executes it or types an error message and aborts. **FIND** is modified by the addition of a call to **SEARCH-CLASS** before searching the **CONTEXT** and **CURRENT** vocabularies if the word

```
ONLY  FORTH  ALSO CLASS@ SINGLE

CLASS FinancialHistory <SUPER Object
CLASS@ FinancialHistory DEFINITIONS

: cashOnHand    (S 'hist -- 'n )                    ;
: incomes       (S 'hist -- 'dict )  2+    @ ;
: expenditures (S 'hist -- 'dict )  4 +   @ ;

: initialBalance    (S 'dict1 'dict2 n -- )
  CREATE  ,  SWAP , , ;
: new                (S 'dict1 '
  CREATE  0 ,  SWAP , , ;

: totalReceivedFrom    (S code hist -- n )
  incomes IntDictionary at ;
: totalSpentFor        (S code hist -- n )
  expenditures IntDictionary at ;

: receive    (S code n hist -- )
  2DUP cashOnHand +!
  SWAP  >R 2DUP totalReceivedFrom  R> +  SWAP
  incomes IntDictionary isAt ;
: spend      (S code n hist -- )
  OVER NEGATE OVER cashOnHand +!
  SWAP  >R 2DUP totalSpentFor  R> +  SWAP
  expenditures IntDictionary isAt ;

FORTH DEFINITIONS

Usage Examples

100 IntDictionary new HouseIncome
100 IntDictionary new HouseExpenses
ONLY  FORTH  ALSO CLASS@ FinancialHistory
HouseIncome HouseExpenses 350 initialBalance Household
utilities 32 Household spend
food 30 Household spend
rent 400 Household spend
wages 1000 Household receive
taxRefund 200 Household receive
Household cashOnHand @ .
```

Figure One
Example Application

is not found in the active class and by the addition of code to set the active class to null at the end of the search process. **SEARCHCLASS** simply follows the class's superclass chain while calling **(FIND)** to search each class's linked list of words along the way.

One possible concern in implementing this proposal is that it introduces another kind of prefix operator to the code-field prefix operators already proposed with multiple-code-field words. One might run into situations where a phrase of the form "CodePrefix ClassName WordName" must be handled. The implementor must ensure that the prefix operators act properly without interfering with each other. One would not want to try to execute the nonexistent second code field of ClassName, for instance. A simple solution would be to implement the code field prefix operators so that they check for intervening class prefix operators or so that the code-field prefix operator sets a system variable which is referred to in determining which code field of a multiple-code-field word to execute; there are many ways that this might be done. It seems logical to require that there be no intervening prefix operators between the class name and the word name.

```
VARIABLE ACTIVECLASS  ( pointer to class to be searched )
VARIABLE NEWCLASS     ( pointer to class being defined )
#THREADS 2* 2+ CONSTANT 'SUPER ( offset to superclass ptr. )

0 ACTIVECLASS !

Class Definition Words

: CLASS   (S -- )
  CREATE  IMMEDIATE  HERE NEWCLASS !
  #THREADS 0  DO  0 ,  LOOP
  HERE  VOC-LINK @ ,  VOC-LINK !  0 ,
  DOES>  ACTIVECLASS ! ;
: <SUPER   (S -- )
  ' >BODY  NEWCLASS @ 'SUPER +  ! ;
: CLASS@   (S -- )
  ' >BODY  CONTEXT ! ;

Class Variables

: CLASSVAR   (S -- )
  CREATE  0 ,
  DOES>  @  ACTIVECLASS ! ;
: CLASS>      (S -- )
  ' >BODY  ' >BODY  ! ;
: (DEFER)     (S -- )
  R> COUNT  2DUP + >R DROP  FIND IF EXECUTE
  ELSE  COUNT TYPE  TRUE ABORT" is undefined." THEN ;
: DEFER       (S -- )
  COMPILE (DEFER)  BL WORD  C@ 1+ ALLOT ; IMMEDIATE

Dictionary Lookup Modifications

: SEARCHCLASS   (S addr -- cfa flag | addr false )
  FALSE  BEGIN
    DUP  ACTIVECLASS @  SWAP 0= OVER AND  WHILE
    DUP 'SUPER + @  ACTIVECLASS !
    SWAP DROP OVER SWAP  HASH @  (FIND)
  REPEAT ;
: FIND           (S addr -- cfa flag | addr false )
  SEARCHCLASS  DUP 0= IF
    ( FIND as defined in F83 )
  THEN  0 ACTIVECLASS ! ;
```

Figure Two
Example Implementation

An Open Question

One of the most difficult questions to answer in the object-oriented programming model concerns the handling of generic classes of composite objects, such as arrays or stacks. How can one efficiently implement a generic array class where subclasses may be simply instantiated for byte arrays, bit arrays, double-length arrays or multi-dimensional arrays of these as they are needed? The solutions I have seen written in Smalltalk seem to be rather inefficient. Charles Moore did not include an **ARRAY** word in his initial design of Forth for basically this reason. I am considering several techniques, but perhaps someone out there already has a solution.

Conclusion

The principal benefit of the proposed approach is that it seems to solve the perceived problems without drastically complicating or changing the present character of Forth. Marriages of Forth and Smalltalk such as Kriya Systems's Neon provide more of Smalltalk's explicit structure at the expense of Forth's flexibility. I find that approach to be overly complex, although I should express my thanks to the implementors of Neon for provoking me to think about this subject. Ultimately, in the author's opinion, the responsibility for the production of elegant, clear and powerful software rests with the programmer. A language should provide a few simple yet powerful and carefully integrated constructs; the discipline and imagination of the programmer provide the rest.